ioProgrammo

QUANDO PHP INCONTRA J2ME
SCRIVI IN JAVA UN'APPLICAZIONE PER IL CELLULARE CHE SFRUTTI I DATI DEL WEB

Rivista + Le Grandi Guide di ioProgrammo n° 3 a € 14,90 in più

VERSIONE PLUS RIVISTA+LIBRO+CD €9,90 — VERSIONE STANDARD RIVISTA+CD €6,90

PER ESPERTI E PRINCIPIANTI

Poste Italiane • Spedizione in a.p. - 45% • art. 2 comma 20/b legge 662/96 - AUT. N. DCDC/033/01/CS/CAL Periodicità mensile • OTTOBRE 2005 • ANNO IX, N.9 (95)

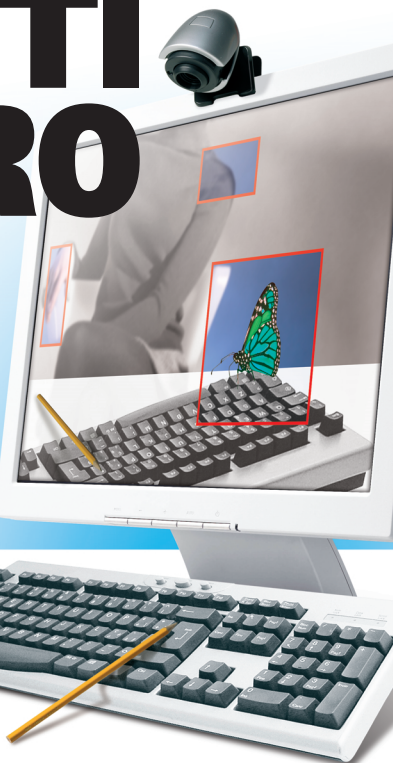
LA TECNICA PIÙ USATA NEI SISTEMI DI VIDEOSORVEGLIANZA

MOVIMENTI SOTTO TIRO

QUALUNQUE COSA SI STIA MUOVENDO IL TUO CODICE LA INDIVIDUA E AGISCE DI CONSEGUENZA!

TEORIA
La motion detection per isolare gli oggetti in movimento non considerando i disturbi video

TECNICA
Quali librerie usare e gli esempi che ti agevolano nella stesura del codice

PRATICA
Acquisire il filmato da una videocamera e tracciare i contorni degli elementi dinamici

INTERNET CAMBIA MARCIA

Ti sveliamo come creare applicazioni dinamiche che non hanno bisogno di ricaricare le pagine

HACKER FUORI DAI SOFTWARE

Ecco come criptare e firmare i dati trasmessi sul web per rendere dura la vita ai malintenzionati

DEBUGGING
CODICE SOTTO STRESS
Impara a testare le applicazioni e consegnarle prive di qualunque bug

JAVA NIO
VELOCI COME FULMINI
Accelera l'accesso al disco rigido e raddoppia le prestazioni del software

VISUAL BASIC
TUTTA LA POTENZA DELLO SCRIPTING
Come fare lavorare insieme Windows scripting Host e VB per gestire facilmente il sistema

NUOVE TENDENZE
ELETTRONICA PER TUTTI!
Crea un programma in C con output su un Display LCD

.NET
PRONTI PER I PLUGIN
Consenti a terze parti di creare estensioni per il tuo software

USARE BENE LA REFLECTION
Il cuore di .NET: codice che analizza e riprogramma se stesso!

VIDEOGAMING
IL PARRALLAX MAPPING
Per dare profondità a immagini altrimenti piatte

JAVA
PROGRAMMARE CON I SALTI
Gli interrupt del codice, quasi come un "GoTo" ma in chiave moderna

UN TIMER DI PRECISIONE
Crea uno scheduler per gestire eventi dipendenti dal tempo

EXTREME PROGRAMMING
SNMP: LO 007 DEI COMPUTER
Ti rivela tutto ma proprio tutto di ogni periferica hardware collegata

I CORSI PER IMPARARE
• ASP.NET • VB .NET 2003

DENTRO IL KERNEL GLI ALGORITMI DA USARE PER EVITARE IL BLOCCO COMPLETO DEL SISTEMA

Anno IX - N.ro 9 (95) - Ottobre 2005 - Periodicità: Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo
http://www.ioprogrammo.it

Direttore Editoriale: Massimo Sesti
Direttore Responsabile: Massimo Sesti
Responsabile Editoriale: Gianmarco Bruni
Redazione: Raffaele del Monaco, Fabio Farnesi
Collaboratori: M. Autiero, D. Bochicchio, L. Buono, F. Cozzolino, E. Del Bono, D. De Michelis, F. Fortino, F. Grimaldi, A. Lacava, F. Lippo, A. Marroccelli, G. Negrelli, A. Panella, F. Paparoni, A. Pelleriti, S. Tanzilli, F. Vaccaro.
Segreteria di Redazione Veronica Longo

Realizzazione grafica: Cromatika S.r.l.
Responsabile grafico: Paolo Cristiano
Coordinamento tecnico: Giancarlo Sicilia
Illustrazioni: M. Veltri
Impaginazione elettronica: Aurelio Monaco

"Rispettare l'uomo e l'ambiente in cui esso vive e lavora è una parte di tutto ciò che facciamo e di ogni decisione che prendiamo per assicurare che le nostre operazioni siano basate sul continuo miglioramento delle performance ambientali e sulla prevenzione dell'inquinamento"

Certificato UNI EN ISO 14001
N. 9191 CRMT

Realizzazione Multimediale: SET S.r.l.
Coordinamento Tecnico: Piero Mannelli
Realizzazione CD-Rom: Paolo Iacona

Pubblicità: Master Advertising s.r.l.
Via C. Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121207
e-mail advertising@edmaster.it
Sales Director: Max Scortegagna
Segreteria Ufficio Vendite: Daisy Zonato

Editore: Edizioni Master S.p.a.
Sede di Milano: Via Ariberto, 24 - 20123 Milano
Tel. 02 831213 - Fax 02 83121330
Sede di Rende: C.da Lecco, zona industriale - 87036 Rende (CS)
Presidente e Amministratore Delegato: Massimo Sesti

ABBONAMENTO E ARRETRATI
ITALIA: Abbonamento Annuale: ioProgrammo (11 numeri) €59,90 sconto 20% sul prezzo di copertina di €75,90
Offerte valide fino al 31/10/05
Costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 02 831212.
La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 02 83121206, oppure via posta a EDIZIONI MASTER via C. Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- cc/p n.16821878 o vaglia postale (inviando copia della ricevuta del versamento insieme alla richiesta);
- assegno bancario non trasferibile (da inviarsi in busta chiusa insieme alla richiesta);
- carta di credito, circuito VISA, CARTASI', MASTERCARD/EUROCARD, (inviando la Vs. autorizzazione, il numero della carta, la data di scadenza e la Vs. sottoscrizione insieme alla richiesta).
- bonifico bancario intestato a Edizioni Master S.p.A. c/o Banca Credem S.p.a. c/c 01 000 000 5000 ABI 03032 CAB 80880 CIN Q (inviando copia della distinta insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.

Sostituzioni: qualora nei prodotti fossero rinvenuti difetti o imperfezioni che ne limitassero la fruizione da parte dell'utente, è prevista la sostituzione gratuita, previo invio del materiale difettato.
La sostituzione sarà effettuata se il problema sarà riscontrato e segnalato entro e non oltre 10 giorni dalla data effettiva di acquisto in edicola e nei punti vendita autorizzati, facendo fede il timbro postale di restituzione del materiale.
Inviare il CD-Rom difettoso in busta chiusa a:
Edizioni Master - Servizio Clienti - Via C. Correnti, 1 - 20123 Milano

Assistenza tecnica: ioprogrammo@edmaster.it

Servizio Abbonati:
tel. 02 831212
@ e-mail: servizioabbonati@edmaster.it

Stampa: Arti Grafiche Boccia S.p.a. Via Tiberio Felice, 7 Salerno
Stampa CD-Rom: Neotek S.r.l.- C.da Imperatore - zona ASI Bisignano (CS)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma

Finito di stampare nel mese di Settembre 2005

Edizioni Master edita: Computer Bild, Idea Web, Go!OnLine Internet Magazine, Win Magazine, Quale Computer, DVD Magazine, Office Magazine, La mia Barca, ioProgrammo, Linux Magazine, Softline Software World, HC Guida all'Home Cinema, MPC, Computer Games Gold, inDVD, I Fantastici CD-Rom, PC VideoGuide, I Corsi di Win Magazine, I Filmissimi in DVD, Filmteca in DVD, La mia videoteca, Win Extra, Home entertainment, Digital Japan, Digital Music, Horror Mania, ioProgrammo Extra, I mitici all'italiana, Win Junior, PC Junior, Guide strategiche di Win Magazine Giochi, Japan Cartoon, I Libri di Quale Computer, Le grandi Guide di Win Magazine, Thriller Mania, Filmteca in DVD, Le Collection

Una finestra sul futuro

Completare un nuovo numero di ioProgrammo pronto per essere consegnato alle edicole costituisce per me, ancora dopo tanto tempo, un'emozione importante. Non è il raggiungimento dello scopo per cui abbiamo lavorato un mese intero a rendermi particolarmente emozionato, ma soprattutto il fatto di avere fra le mani un numero nella sua interezza, stracolmo di tecniche innovative, di informazioni che costituiscono il cuore pulsante della tecnologia. Rileggo ogni articolo con la faccia stupita di chi scopre di volta in volta una nuova visuale verso il futuro. Perché è questo che ioProgrammo è: un'insieme denso di tecnologie che nel corso del tempo vedremo applicate alla vita quotidiana, che ora in qualche caso possono apparire astruse o astratte e che invece fra non molto ci sembreranno usuali e conosciute. Ciascuno di questi articoli è uno spunto per la fantasia, pronto per lasciare che la nostra immaginazione produca quelli che saranno i software di domani. E' il caso della programmazione Embededd , ma in questo numero anche di Ajax ad esempio. Soprattutto Ajax appare ancora lontano dall'essere applicato all'interezza del web, eppure contiene in se tutta la forza per attuare una rivoluzione impensabile solo fino a qualche anno fa. Se avrete la pazienza di leggere l'articolo e lasciare viaggiare la fantasia sarete in grado di immaginare come sarà il Web di domani e soprattutto di iniziare a costruirlo da subito e con le vostre mani.

Fabio Farnesi

All'inizio di ogni articolo, troverete un simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre \soft\codice\ e \soft\tools\) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it.

MOVIMENTI SOTTO TIRO

QUALUNQUE COSA SI STIA MUOVENDO IL TUO CODICE LA INDIVIDUA E AGISCE DI CONSEGUENZA!

- Isolare gli oggetti in movimento non tenendo conto dei disturbi del video
- Quali librerie usare e gli esempi che ti agevolano nella stesura del codice
- Acquisire il filmato da una videocamera e tracciare i contorni degli elementi dinamici

pag. 16

HACKER FUORI DAI SOFTWARE

Ecco come criptare e firmare i dati trasmessi sul web per rendere dura la vita ai malintenzionati pag. 46

http://forum.ioprogrammo.it

QUALCHE CONSIGLIO UTILE

I nostri articoli si sforzano di essere comprensibili a tutti coloro che ci seguono. Nel caso in cui abbiate difficoltà nel comprendere esattamente il senso di una spiegazione tecnica, è utile aprire il codice allegato all'articolo e seguire passo passo quanto viene spiegato tenendo d'occhio l'intero progetto. Spesso per questioni di spazio non possiamo inserire il codice nella sua interezza nel corpo dell'articolo. Ci limitiamo a inserire le parti necessarie alla stretta comprensione della tecnica.

RIVISTA + LIBRO + CD-ROM in edicola

RIVISTA + CD-ROM in edicola

Prodotti del mese

Instant Developer

Instant Developer apre la strada ad una nuova tipologia di software di sviluppo per Web Application

Instant Developer si avvicina a quello che normalmente viene definito come ambiente RAD ereditandone la capacità di creare applicazioni basate su componenti, sul drag & drop, e sulla modalità di "disegno dell'applicazione" tipica degli ambienti RAD.
Pro Gamma lo definisce "il primo sistema di sviluppo relazionale al mondo" perché è in grado di mantenere automaticamente integrate tutti le parti dell'applicazione, cosa che è a carico dei programmatori nei tradizionali ambienti. Si tratta di un software incredibilmente utile che coniuga un'alta scalabilità a un concetto di sviluppo del tutto nuovo per la programmazione web

[pag.91]

Snort 2.33

Il sistema che rileva gli intrusi

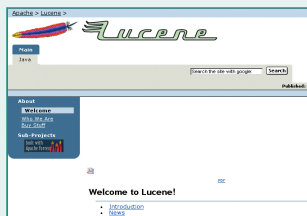
Snort è un Intrusion Detection System, per gli amici IDS. Ovvero un sistema che si mette in ascolto sulla vostra scheda di rete e rileva tutti i pacchetti che vi passano attraverso. Ogni pacchetto viene confrontato con una serie di regole. E se qualche pacchetto viene identificato come potenzialmente pericoloso viene generato un Alert che segnala all'amministratore che c'è la possibilità che un qualche attacco sia in corso verso il proprio sistema.
Un vero 007 che non lascia scampo a chi cerca di utilizzare in modo improprio la potenza di TCP/IP ma anche di sfruttare le enormi falle che ogni giorno vengono scovate all'interno dei protocolli e dei software più comuni

[pag.107]

Lucene 1.4.3

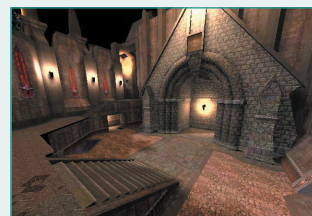
Un segugio per le ricerche

Prima Google ha rilasciato la sua Desktop Search Bar che consentiva di indicizzare il contenuto del vostro HD per potere effettuare delle ricerche veloci. Un po' come il famoso motore di ricerca, ma locale al vostro PC. Subito dopo si sono scatenati, ovviamente i concorrenti OpenSource. Lucene è uno di questi. Sulla sua base è nato ad esempio il progetto Beagle, uno strumento scritto in Mono che sfrutta lucene per creare un proprio motore di ricerca. Tocca a voi adesso migliorare beagle, oppure utilizzare lucene all'interno dei vostri software. Noi aspettiamo solo che ci mandiate le vostre soluzioni. La tecnica è davvero interessante e stimola la fantasia per quanto riguarda i tool di sviluppo.

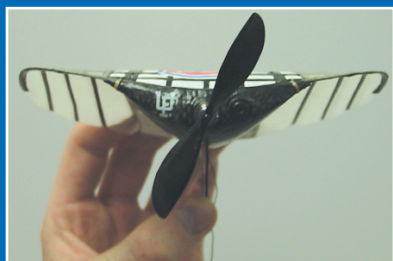
[pag.106]

Irrlicht 0.1.11

Il motore 3D per lo sviluppo di VideoGames

Da quando abbiamo cominciato a parlare di Irrlicht sulle pagine di ioProgrammo è stata un escalation di diffusione. Il merito è sicuramente la potenza e la facilità d'uso di questo straordinario Engine 3D. In questo numero di ioProgrammo ne abbiamo parlato in relazione all'applicazione di effetti di parallax mapping su sequenze animate. In condizioni normali i calcoli per l'applicazione del parrallax mapping sarebbero piuttosto complessi, è qui che si vede la potenza di Irrlicht che mette a disposizioni poche semplici primitive per la realizzazione di questo effetto.
Irrlicht ha davvero tutti i numeri per diventare un punto di riferimento in questo settore.

[pag.106]

News

IN ARRIVO I ROBOT CONTROLLORI DEL CIELO

In uno scenario da film di fantascienza se vi capitasse di vedere un robot delle dimensioni di qualche centimetro solcare i cieli probabilmente vi lascereste sfuggire un'esclamazione di stupore per poi elogiare la fantasia di regista e sceneggiatore. Eppure questo scenario potrebbe attraversare il confine della fantascienza per giungere nella realtà. Gli scienziati del dipartimento di meccanica aerospaziale della Florida stanno appunto lavorando a un progetto di videosorveglianza dei cieli che vedrebbe l'impiego di minirobot di dimensioni che variano da 6 a 24 pollici che simulano in tutto e per tutto il comportamento degli uccelli utilizzando una tecnica di "metamorfosi dinamica" che consente di mimare in modo quasi realistico il comportamento naturale del volo degli animali. Il Governo Federale sembrerebbe seriamente intenzionato ad adottare questa tecnica per la videosorveglianza aerea delle grandi città.

RILASCIATO GOOGLE DESKTOP SEARCH 2.0

Il motore di ricerca più famoso al mondo da qualche tempo sta investendo in tecnologie che pur coinvolgendo in un qualche modo la ricerca e l'indicizzazione di informazioni, esulano dalla classica ricerca sul Web. È il caso di Google Desktop Search, che consente di indicizzare con una tecnologia molto simile a quella usata sul web documenti di ogni tipo locali al proprio computer. L'applicazione ha riscosso un notevole successo. D'altra parte non capita raramente di dover ricercare all'interno del proprio computer un documento o un'immagine per poi ritrovare l'informazione che ci serviva all'interno di una email magari. GDS risolve questo problema in modo elegante e soprattutto con un'applicazione che fa della velocità di ricerca uno dei suoi punti di forza. La versione 2.0 di GDS include diverse novità tutte di notevole interesse. Oltre a un migliorato supporto per la ricerca all'interno delle Email che coinvolge sia la classica posta elettro-

QUASI PRONTO INTERNET EXPLORER 7

Al momento in cui scriviamo la prima beta pubblica ufficiale del browser di Microsoft non è ancora stata resa disponibile. La data dichiarata per il rilascio di questa nuova versione fa riferimento a un generico "Settembre 2005". Diverse sono le novità che dovrebbero coinvolgere IE7, alcune di carattere commerciale altre di carattere squisitamente tecnico. Dal punto di vista strettamente commerciale sembrerebbe che il nuovo prodotto non sarà reso disponibile per la piattaforma Apple ma unicamente per Windows XP/2003 e per il prossimo Windows Vista. Sono previsti piccoli ritocchi anche allo storico logo. Dal punto di vista strettamente tecnico in IE7 sono previste novità per quanto riguarda l'Antiphishing ed il browser sarà finalmente Tabbed ricalcando le orme dei concorrenti Firefox ed Opera. Dal punto di vista della sicurezza, la novità più interessante sembrerebbe consistere nel passaggio dalla normale tecnologia di parsing URI che prevede un parsing diretto delle stringhe contenute in URL, ad una tecnologia CURI che prevede l'uso di un oggetto specifico che metterebbe a disposizione metodi e proprietà per accedere agli URL e verificarne la consistenza.

RSS3 AI NASTRI DI PARTENZA

Really Simple Syndication è questo l'acronimo di RSS. Per intenderci si tratta della tecnologia XML che consente di scambiare flussi di informazioni tramite uno standard riconosciuto ed elaborato grazie alla tecnologia XML. RSS è diventato nel tempo un riferimento soprattutto in Internet dove viene costantemente utilizzato per la condivisione delle notizie o per trasferire contenuti informativi da un sito all'altro. Nei mail reader più recenti molto spesso viene ormai integrato anche un parser di documenti RSS che consente di ricevere un flusso costante di informazioni dai siti che adottano questo standard.
Data la rilevanza che RSS sta assumendo in campo tecnologico, appare evidente come il conformarsi a questa terza draft dello standard sia di fondamentale importanza.
Al momento la bozza è stata resa disponibile all'indirizzo *www.rss3.org/rss3lite.html#spec_introduction*. Si tratta di una "First Draft" ancora lontana dall'entrare in circuiti di produzione, tuttavia si nota già nella prima bozza la volontà di volere fare ordine in un settore in forte espansione, riassumendo in unica versione tutte le caratteristiche delle cinque release precedenti e delle varie derivazioni a cui hanno dato luogo, compresi i vari atom feed e le altre, proposte in maniera più o meno disordinata da una pletora di soggetti compresi in una qualche misura Microsoft e Google.

nica aggiungendo dei filtri di ricerca per Outlook, è anche possibile indicizzare documenti provenienti da Gmail con una maggiore integrazione con il servizio di posta offerto dallo stesso Google. La maggiore innovazione sembrerebbe però la Sidebar verticale che racchiude in un unico strumento i numerosi servizi offerti da GDS. Nella stessa Sidebar è ora inclusa una funzionalità che consente di leggere i flussi RSS, inoltre è stato aggiunto un supporto verso Microsoft Office che consente di interagire in modo più efficace con i blog che usufruiscono della piattaforma blogger.com

PROBLEMI DI VULNERABILITÀ PER ACROBAT

Che i documenti in formato PDF siano ormai uno standard quanto lo può essere un file di testo è assodato. Non c'è utente al mondo che non abbia almeno una volta aperto un file PDF. Fino ad ora non si erano registrate particolari vulnerabilità relative allo standard di Adobe, e tuttavia l'impegno degli hacker nello sfruttare ogni minima possibilità per scovare falle di sicurezza nei software più diffusi è davvero senza sosta. Questa volta è il turno di Acrobat Reader. Di fatto imbattendosi in un documento formato in maniera particolare sul Web è possibile sfruttare il buffer overflow e mandare in crash l'applicazione aumentando il rischio di esecuzione di codice pericoloso.
Adobe è prontamente corsa ai ripari, così dalla versione 5.0 è necessario upgradare alla versione 5.2, dalla 6.0 alla 6.0.4 e dalla versione 7 alla 7.0.3. Attualmente i rischi legati a questa vulnerabilità sono bassi, tuttavia è importante procedere prima possibile ad effettuare l'upgrade.

DISPONIBILI I SORGENTI DI QUAKE III ARENA

Non sarebbe la prima volta che idSoftware rende disponibili in forma GPL i sorgenti dei propri prodotti.
Era già successo per Doom e ora è il turno di Quake III Arena.
La notizia è di quelle che scottano, sia perchè Quake III così come Doom è di quei giochi che hanno segnato la storia, sia per quantità di innovazioni apportate, sia per tecnologia utilizzata.
Poter studiare, modificare e personalizzare questi sorgenti rappresenta una grande risorsa per chi ama questo genere di programmazione.
L'esperienza inoltre insegna che rendere disponibili i sorgenti aumenta anche il numero degli sviluppatori, il che generalmente porta come conseguenza un grande flusso di informazioni e migliorie che contribuiranno sicuramente a creare videogiochi sempre più divertenti e realistici.

PC, VENDITE IN RIPRESA MA RENDITE IN CALO

A fine 2005 il mercato della vendita di personal computer dovrebbe fare segnare un 12.7 per cento in più rispetto al 2004. Se da un lato l'aumento delle vendite prelude ad una ripresa di un mercato che ha fino ad ora marciato a ritmi blandi, d'altro canto i produttori non sembrano beneficiare di questo trend. I guadagni fatti registrare dall'industria dell'hardware aumenterebbero appena di uno 0.5 per cento a causa del taglio dei prezzi resosi necessario per alimentare un mercato al ribasso. Nel 2006 si prevede un ulteriore aumento delle vendite nella misura di un 10.5 per cento e tuttavia ancora una volta i guadagni dei produttori dovrebbero aumentare appena di uno 0.4 per cento.
A fare da traino al mercato sono e continueranno ad essere anche per il prossimo anno i notebook e le periferiche wireless, che grazie a prezzi contenuti e comodità di utilizzo si rendono piuttosto appetibili per gli utenti. La ricerca è stata condotta da Gartner, gruppo leader nel mondo per le ricerche nel mercato IT.

Quale debugger usare?

Buongiorno, mi chiamo Diego e con la mia azienda stiamo sviluppando un software per il disegno meccanico. Lo sviluppo procede piuttosto bene, ma il software si dimostra molto complesso ed abbiamo raggiunto ormai svariati milioni di righe di codice. Eccetto i normali bug, o difetti di programmazione che nel tempo procediamo a correggere tramite test tradizionali, abbiamo un problema relativo all'occupazione della memoria. Cioè l'applicativo consuma risorse eccessive di memoria. Non riusciamo a debuggare a sufficienza l'applicazione per capire dove abbiamo commesso errori di programmazione. Il problema fondamentale è che non abbiamo ancora trovato un sistema abbastanza leggero da consentirci di debuggare un software di qualche milione di righe e contemporaneamente, sufficientemente potente da poterci restituire informazioni affidabili. Qualche consiglio?

Diego da Albissola

Gentile Diego proprio di recente un caso simile si è verificato per un'azienda di Bologna: Think3. È una multinazionale ormai con circa 5000 clienti e 400 dipendenti. Il loro focus è proprio quello dello sviluppo di software per la progettazione meccanica. Uno dei loro software di punta è Thinkdesign, che è anche uno dei prodotti storici, come certamente saprà, in questo settore. Uno dei problemi di Think3 nello sviluppare i loro prodotti era proprio legato all'occupazione di memoria che in alcuni casi faceva degradare le prestazioni e in qualche caso creava problemi di crash dell'applicativo. Dopo vari tentativi di debugging, probabilmente molto simili a quelli che ha già tentato la sua azienda, Think3 ha risolto il problema utilizzando un software sviluppato da Compuware: *BoundsChecker.* Si tratta di uno strumento piuttosto interessante. BoundsChecker consente di evitare l'"instrumentazione" cioè evita di dovere ricompilare tutte le volte l'applicativo per poterlo analizzare, e questo nel caso di software composto da qualche milione di righe di codice e molti moduli esterni è sicuramente un grande vantaggio. Inoltre grazie ad alcuni moduli esterni di DevPartener studio è possibile verificare il livello delle prestazioni raggiunte dal software analizzato. In Think3 garantiscono che l'uso di BoundsChecker ha consentito di abbattere totalmente il problema dell'occupazione di memoria e conseguente degrado delle prestazioni dei loro software. Consiglierei di provare BoundsChecker scaricando una trial version seguendo il link *"try"* dall'indirizzo *http://www.compuware.com/products/devpartner/bounds.htm*. Controlli se con questo tool di CompuWare la sua azienda riesce a risolvere il problema.

Che differenza c'è fra Imap4 e Pop3?

Nel configurare la posta elettronica mi capita spesso di dovere scegliere fra Pop3 ed imap4. Ho capito che Pop3 è il servizio che mi consente di prelevare la posta da un server, ma che differenza c'è con Imap? quando viene utilizzato l'uno e quando l'altro?

Valentina da Catanzaro

Come hai ben ricordato, Pop3 è un servizio che consente di scaricare la posta. Una sorta di scatola all'interno della quale la posta arrivata sul server SMTP viene spostata tramite un *Mail Transfer Agent* – MTA in attesa che un client di posta elettronica possa scaricarla in locale. Questo tipo di strumento è utile, ma presenta alcuni limiti. Supponiamo ad esempio che sia tu che un tuo collega dobbiate accedere alla stessa casella di posta, ad esempio quella relativa all'assistenza clienti. Se tu scaricassi la posta in locale il tuo collega non troverebbe nessun messaggio sul server. Se decideste di lasciare la posta sul server sarebbe impossibile sapere chi dei due ha risposto ad un messaggio a meno che non lo comunichiate verbalmente l'uno all'altro. Allo stesso modo non rimarrebbe traccia della risposta, quindi nessuno dei due potrebbe sostituire l'altro in caso di emergenza. IMAP4 è un protocollo che nelle intenzioni doveva sostituire POP3, e fornire delle soluzioni anche per questo tipo di problema. Nella realtà si tratta di un protocollo poco utilizzato se non nelle realtà aziendali dove realmente può apportare diversi vantaggi. IMAP prevede che la posta rimanga centralizzata sul server, e prevede che i client che vi accedano possano condividere le informazioni relative alla posta elettronica. Se uno dei due marca un messaggio come "risposto" l'altro ne avrà visione. Inoltre IMAP consente ad esempio la pubblicazione delle cartelle a favore di altri utilizzatori della stessa gerarchia di posta, oppure dei contatti. Insomma si tratta di un protocollo più lento, complesso e macchinoso di POP3, ma che esporta alcuni metodi che per certi versi rendono la posta elettronica simile ad una directory condivisa ma con qualche funzione in più. Si tratta di un metodo utile da utilizzare in reti aziendali che svolgano tramite posta elettronica anche funzioni di gestione della produzione.

Migrare da MySQL a PostgreSQL

Salve, da qualche tempo ho iniziato a studiare PostgreSQL. Lo trovo estremamente interessante e ricco di funzioni aggiuntive rispetto a MySQL che è il database che attualmente uso. Ora, il punto è che mi piacerebbe iniziare a migrare i miei siti da mySQL a postgreSQL o quantomeno vorrei fare qualche esperimento per capire se è un'operazione che realmente possa essermi utile oppure no. C'è un modo sem-

plice per effettuare questa migrazione? È un'operazione conveniente? cosa mi consigliate di fare?
Dario da Cagliari

Dario, mySQL è comunque un grande database. Non vorremmo entrare qui nel dettaglio del confronto ed aprire una polemica solitamente sterile su quale sistema sia migliore. Ambedue presentano vantaggi e svantaggi. Certamente il numero di caratteristiche esposte da PostgreSQL è elevato, l'affidabilità è alta, ed il database è incredibilmente stabile. È conveniente effettuare la migrazione nella misura in cui allo stato attuale lavorare con mySQL ti comporti qualche problema. In particolare le stored procedure e le view sono un buon motivo per passare a postgreSQL. Se invece il confronto si basa esclusivamente sulle prestazioni non è forse utile effettuare la migrazione. Per quanto riguarda gli aspetti esclusivamente tecnici. La prima operazione da compiere è effettuare un dump delle proprie tabelle mySQL:

```
mysqldump -u username -p nome_db
                        > nome_db.sql
```

a questo punto bisogna intervenire sulle istruzioni di creazione delle tabelle per verificare la compatibilità dei tipi. Un elenco completo dei tipi supportati da postgreSQL è reperibile all'indirizzo: *http://www.postgresql.org/docs/8.0/interactive/datatype.html*. In particolare bisogna porre attenzione alle tabelle che fanno uso di campi autoincrementali. PostgreSQL gestisce i campi incrementali attraverso l'uso delle sequenze. Una sequenza è una tabella che mantiene un indice del valore degli incrementi di un campo dichiarato in una tabella ad essa correlato.
Tanto per fare un esempio pratico:

```
CREATE SEQUENCE tablename_colname_seq;
CREATE TABLE tablename (colname integer
  DEFAULT nextval('tablename_colname_seq')
                              NOT NULL);
```

Si vede come il campo colname sia legato alla sequenza gestita dalla tabella *tablename_colname*. Un particolare tipo di dato chiamato "SERIAL" implementa automaticamente il meccanismo di cui sopra. Per cui tutti i campi autoincrementali di mySQL devono essere dichiarati SERIAL in PostgreSQL. Se si usa il tipo SERIAL invece che la sua espansione come mostrato negli esempi precedenti, si deve tenere conto che le tabelle di sequenza generate in automatico saranno nominate come *tablename_colname_seq*. Una volta terminato di rendere omogenee le tabelle ai nuovi tipi di dato supportati da PostgreSQL puoi provvedere a ricrearle all'interno del nuovo db. Importare i dati è meno complicato. In teoria l'unica differenza fra i due db dovrebbe consistere nell'uso delle virgolette. Le virgolette vanno protette in PostgreSQL con uno \. Fatto questo puoi provare a importare i dati in PostgreSQL. Sicuramente qualcosa andrà storto, bisogna verificare il singolo caso e capire quale problema potrebbe essere accaduto. Restano poi da adeguare gli script PHP al nuovo db.
La migrazione non sembra essere un'operazione complessa, tuttavia affinché tutto funzioni senza intoppi è necessaria una certa attenzione e un po' di ore di lavoro, per cui si tratta di un'operazione da fare con calma, e sicuramente da testare prima di porre in esecuzione in sistemi in produzione.

A chi spedire la posta?

Alla nostra redazione arriva spesso un considerevole numero di email riguardante temi specifici. Per consentirci di rispondere velocemente e in modo adeguato alle vostre domande abbiamo elaborato una FAQ – Frequently Ask Question – o risposte alle domande frequenti in italiano, di modo che possiate indirizzare le vostre richieste in modo mirato.

Problemi sugli abbonamenti

Se la tua domanda ha a che fare con una delle seguenti:

- Vorrei abbonarmi alla rivista, che devo fare?
- Sono un abbonato e non ho ricevuto la rivista, a chi devo rivolgermi?
- Sono abbonato ma la posta non mi consegna regolarmente la rivista, a chi devo rivolgermi?

Contatta **abbonamenti@edmaster.it** specificando che sei interessato a ioProgrammo. Lascia il tuo indirizzo email e indica il numero dal quale vorresti far partire l'abbonamento. Verrai contattato al più presto. Oppure puoi chiamare lo **02 831212**

Problemi sugli allegati

Se riscontri un problema del tipo:

- Ho acquistato ioProgrammo ed il Cdrom al suo interno non funziona. Chi me lo sostituisce?
- Ho acquistato ioProgrammo ma non ho trovato il cd/dvd all'interno, come posso ottenerlo?
- Vorrei avere alcuni arretrati di ioProgrammo come faccio?

Contatta **servizioclienti@edmaster.it**
Non dimenticare di specificare il numero di copertina di ioProgrammo e la versione: con libro o senza libro. Oppure telefona allo **02 831212**

Assistenza tecnica

Se il tuo problema è un problema di programmazione del tipo:

- Come faccio a mandare una mail da PHP?
- Come si instanzia una variabile in c++?
- Come faccio a creare una pagina ASP.NET

o un qualunque altro tipo di problema relativo a tecniche di programmazione, esplicita la tua domanda sul nostro forum: **http://forum.ioprogrammo.it**, uno dei nostri esperti ti risponderà. Le domande più interessanti saranno anche pubblicate in questa rubrica.

Problemi sul codice all'interno del CD

Se la tua domanda è la seguente

- Non ho trovato il codice relativo all'articolo all'interno del cd

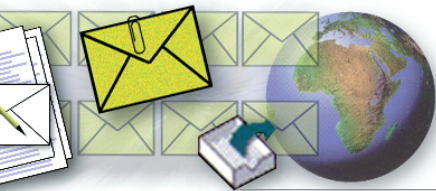
Consulta la nostra sezione download all'indirizzo **http://cdrom.ioprogrammo.it**, nei rari casi in cui il codice collegato ad un articolo non sia presente nel cdrom, senza dubbio verrà reso disponibile sul nostro sito.

Idee e suggerimenti

Se sei un programmatore esperto e vuoi proporti come articolista per ioProgrammo, oppure se hai suggerimenti su come migliorare la rivista, se vuoi inviarci un trucco suggerendolo per la rubrica tips & tricks invia una email a **ioprogrammo@edmaster.it**

Che cosa vuol dire Promiscuous Mode?

Buongiorno, ho letto su Internet che c'è una particolare modalità con cui la scheda di rete può essere configurata, denominata "Promiscuous Mode". Non ho ben capito a cosa serve e come funziona. Mi date qualche indicazione?
Antonio da Napoli

Buongiorno Antonio. Le schede di rete sono dotate di un particolare indicatore chiamato "Mac Address". I pacchetti in circolo ad esempio all'interno di una Lan vengono confrontati con il Mac Address della scheda. Se l'indirizzo Mac presente nel pacchetto coincide con quello della scheda, il pacchetto viene inviato alla CPU che lo utilizza secondo i metodi dello stack TCP/IP.
Questo serve a fare in modo che solo i pacchetti indirizzati ad una particolare scheda siano effettivamente utilizzati. Settando la scheda di rete in modo promiscuo viceversa tutti i pacchetti vengono indirizzati verso la CPU e questo consente in un certo qual senso di controllare la rete.

Direttamente dal forum di ioProgrammo http://forum.ioprogrammo, le discussioni più "Hot" del momento

Visual Basic .NET

Aprire file doc con

Ciao a tutti!!!Sto imparando da autodidatta visual basic .net (ho iniziato da pochissimo). Mi servirebbe il codice per aprire un file .doc già esistente se un checkbox è selezionato. Ho iniziato così:

```
If CheckBox1.Checked = True Then...
```

ma mi manca tutto il resto! Scusate la domanda senz'altro banalissima (ma del resto ho iniziato solo da 2 settimane) ringrazio anticipatamente, spero che mi rispondiate al più presto!!!

Tony81

http://forum.ioprogrammo.it/thread.php?threadid=6272&boardid=27

Risponde amdbook

...puoi fare in questo modo:

```
Dim pr As System.Diagnostics.Process = Nothing
pr = System.Diagnostics.Process.Start(
        "percorso_completo_file_doc")
```

Assembly

Interrupt e Win 32

Mi dilettavo, da vero principiante in asm nello scrivere, usando la direttiva asm, hello world da un'applicazione VC++. Questo è il codice che sto provando ad inserire, ma che genera 2 errori in compilazione:

```
__asm mov ax,cs;
__asm mov ds,ax;
__asm mov ah,9;
__asm mov dx, offset Hello;
__asm int 21h;
__asm xor ax,ax;
__asm int 21h;
__asm Hello: ;
__asm db "Hello World!",13,10,"$" ;
```

Qualcuno gentilmente, mi viene in aiuto? Gli errori sono 2, almeno per ora:

1) L'*opcode int* viene riconosciuto come tipo di dato del C++.

2) *__asm mov dx, offset Hello;* Mi dice che gli operatori devono essere di uguale grandezza.

hyde

http://forum.ioprogrammo.it/thread.php?threadid=6299&boardid=20

Risponde Salvatore Meschini

Le chiamate dirette di interrupt non sono lecite in ambienti a 32 bit. Windows, per esempio, utilizza NTVDM (NT Virtual DOS Machine) per emulare il sottosistema DOS (in realtà l'emulatore è rappresentato dal file ntdos.sys). Morale della favola? O rinunci alle chiamate INT oppure utilizzi un compilatore a 16 bit. Esistono delle alternative ma sono assolutamente dirty! Compilatori che producono codice a 16 bit e che quindi sono utilizzabili sono:

Open Watcom *http://www.digitalmars.com/download/freecompiler.html"\t "_blank"*

Digital Mars *http://bdn.borland.com/museum/" \t "_blank" BC++ 2*

C#

Drag & Drop da una Picture Box

**Ciao a tutti, devo realizzare un progetto che implementi il gioco della torre di Hanoi. Ho problemi sull'interfaccia grafica. Devo riuscire a trascinare un'immagine da una Picture Box ad un'altra picture box. È da un bel po che ci sbatto la testa ma non ne vengo a capo.
Per cortesia aiutatemi**

kunio

http://forum.ioprogrammo.it/thread.php?threadid=6305&boardid=29

Risponde AmdBox

..il controllo PictureBox non ha la proprietà "AllowDrop", la quale impostata su "True" permette la gestione del "Drag&Drop", per sopperire a tale "mancanza" puoi incapsulare il controllo PictureBox all'interno di un controllo Panel (impostando la proprietà Dock del PictureBox su Fill in modo che il Panel venga "riempito" in automatico anche durante il resize). Per il codice d'esempio è necessario inserire in una Windows Form due controlli PictureBox, rispettivamente PictureBox1 e PictureBox2, il secondo inserito all'interno di un controllo Panel (Panel1) come detto in precedenza.
Lo scopo dell'esempio "trascinare" l'immagine dal controllo PictureBox1 al controllo PictureBox2 (naturalmente per il contrario è possibile

eseguire la procedura a specchio), bando alle chiacchere ecco il codice:

```
private void Form1_Load(object sender,
                    System.EventArgs e)
{
this.panel1.AllowDrop = true;
this.pictureBox1.SizeMode =
         PictureBoxSizeMode.StretchImage ;
this.pictureBox2.SizeMode =
PictureBoxSizeMode.StretchImage ;
}
private void pictureBox1_MouseMove(
     object sender, System.Windows.Forms
                  .MouseEventArgs e)
{
if(e.Button == MouseButtons.Left )
{
this.pictureBox1.DoDragDrop
              (this.pictureBox1.Image
             ,DragDropEffects.Copy );
}
}
private void panel1_DragOver(object sender,
  System.Windows.Forms.DragEventArgs e)
{
if(e.Data.GetDataPresent
     (System.Windows.Forms.DataFormats
                        .Bitmap ))
{
e.Effect = System.Windows.Forms
              .DragDropEffects.Copy ; }
}
private void panel1_DragDrop(object sender,
  System.Windows.Forms.DragEventArgs e)
{
if(e.Data.GetDataPresent (System.
     Windows.Forms.DataFormats.Bitmap ))
{
this.pictureBox2.Image =
              this.pictureBox1.Image ;
this.pictureBox1.Image = null; }
}
```

Manipolare le stringhe

Ciao. Mi ritrovo a dover trattare delle stringhe in c#, devo scrivere una cosa del genere:

```
Dim Str as string = "Prova";
```

Devo concatenare la stringa:

```
Str &= controlschar.cr
```

oppure

```
Str &= Chr(13)
```

In c# non so come fare queste due semplici cose: concatenare una stringa e soprattutto ottenere il carattere Cr di ritorno a capo. Scusate la domanda un po semplice ma proprio non so come si fa, mi date una mano?

AleTax

http://forum.ioprogrammo.it/thread.php?threadid=6289&boardid=29

Risponde Cteniza

Visto che le stringhe sono delle entità immutabili ti consiglio di utilizzare sempre la stringbuilder.

Esempio:

```
Dim sb As New System.Text.StringBuilder
sb.Append("pippo")
sb.Append(10)
sb.Append("x"C)
dim myString As String = sb.ToString()
```

Java

Monitorare il puntatore del mouse

Salve a tutti, qualcuno mi saprebbe dire come far si che il puntatore del mouse invece di quello che normalmente si vede sia un'immagine. È possibile vero?
Io ho una classe in cui vi è il main, un altra in cui vi è un jframe e una ancora in cui vi è un jpanel...

esponenzial

http://forum.ioprogrammo.it/thread.php?threadid=6301&boardid=18

Risponde giamig

Prova il seguente codice

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
public class MioFrame extends JFrame {

public Pannello_intro pannello;
Container contentPane = getContentPane();

ImageIcon immTemp;
Image immagine;
Cursor cursore;

public MioFrame(){

immTemp= new ImageIcon(
                    "./immagineCursore.gif");
immagine = immTemp.getImage();
cursore = Toolkit.getDefaultToolkit()
          .createCustomCursor(immagine,new
                    Point(0,0),"myCursor");

this.setCursor(cursore);
setTitle("ciao");
setSize(1024, 76);
setLocation(0,0);
setDefaultCloseOperation(EXIT_ON_CLOSE);
pannello = new introduzione();
setContentPane(panelGioco);
}
}
```

con questo codice quando il mouse sarà sul frame assumerà la forma del nuovo cursore.

Convertire da Int a string, è possibile?

Ciao, Il mio problema è che ho una variabile di tipo int (che chiamo t) calcolata dal mio programma, questa devo inserirla in una label nell'interfaccia.
Facendo *label1.Text=t*; ovviamente mi da errore perché non posso inserire un int in un oggetto che si aspetta una stringa. Facendo *label1.Text= (string)t;* mi dice che non si può convertire un int in una stringa.
C'è un modo per aggirare il problema?

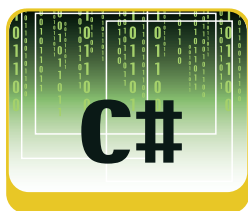
http://forum.ioprogrammo.it/thread.php?threadid=6337&boardid=29

Risponde Kunio

Tux usa il metodo *ToString()*, quindi:

```
label1.Text=t.ToString();
```

Motion Detection Algorithms

Esistono diverse tecniche per "catturare" i movimenti di oggetti in una sequenza video. In questo articolo mostreremo come lavorare con filmati in "presa diretta" da una videocamera o da una webcam

Il problema della motion detection in realtime consiste nell'analizzare un segnale video man mano che viene acquisito dal computer, in modo da individuare immediatamente qualsiasi "variazione" di contenuto all'interno di una ripresa. La sorgente video può essere campionata da ogni dispositivo interfacciabile direttamente al PC (WebCam anche over IP o videocamera digitale connessa tramite firewire) oppure attraverso scheda di acquisizione analogica/digitale (telecamere di sorveglianza di solito collegate ad un videoregistratore). Naturalmente si ha anche la possibilità di analizzare un filmato già esistente (in questo caso si ha un'elaborazione posticipata). Il campo di applicazione di questa tecnica è quindi molto vasto:

Controllo del traffico
Una videocamera installata lungo un tratto della rete stradale, o presso un incrocio, consente un monitoraggio immediato e costante della viabilità favorendo eventuali operazioni di soccorso o intervento sul traffico, così come permette all'utente privato di valutare la situazione del traffico lungo il percorso prescelto e di scegliere eventuali percorsi alternativi. L'adozione di sistemi di motion detection consente di effettuare analisi dettagliate su molte informazioni come ad esempio il numero di autoveicoli che attraversano un tratto stradale o le traiettorie seguite durante il loro moto oppure il superamento di zone interdette al traffico urbano.

Fig. 1: Monitoraggio del traffico mediante un'applicazione della tecnica "motion detection"

Assistenza Disabili
Un'altra applicazione molto interessante è l'individuazione del movimento oculare in pazienti impossibilitati al movimento e alla parola. Per queste persone, la motion detection può rappresentare l'unico mezzo per poter comunicare con gli altri: ad esempio può consentire la composizione di parole trasformando il movimento degli occhi nel movimento di un cursore su di uno schermo.

Videosorveglianza
Non bisogna però dimenticare da dove sono partiti tutti gli studi della motion detection: la videosorveglianza. Visionare lunghissime riprese video di un'area sorvegliata spesso vuol dire stare molte ore di fronte ad un monitor lavorando spesso di fermoimmagine. Per facilitare l'analisi delle sequenze video si sono sviluppati sistemi automatizzati di estrapolazione per le singole scene ritenute "interessanti" in base ad alcuni parametri preimpostati. La naturale applicazione della motion detection è quindi quella del controllo continuo nel tempo sugli accessi nelle aree riservate, oppure per monitorare il movimento dei clienti all'interno di un supermercato in modo da determinare i percorsi usati per effettuare la spesa.

In questo articolo utilizzeremo la tecnica della *motion detection* per costruire un'applicazione che analizza la sequenza di un filmato avi e traccia i contorni delle aree che hanno subito una differenza rispetto ai frame immediatamente precedenti. Ad esempio il movimento di un uomo che cammina all'interno di una stanza sarà evidenziato da un contorno rosso. Se avete già fatto funzionare la fantasia non ci vuole molto a ipotizzare che un'applicazione

del genere potrebbe essere estesa per vettorializzare il movimento partendo da una sequenza reale…

TECNICHE DI MOTION DETECTION

Esistono diverse soluzioni per rilevare il movimento di oggetti in sequenze video, il che rappresenta il primo passo per estrarre informazioni utili. In generale il problema di elaborazione dei filmati viene affrontato utilizzando algoritmi che si basano sulla ricerca delle "differenze", come in alcuni giochi di enigmistica: da una sequenza vengono estratti e messi a confronto due frame (cioè singole immagini) successivi, determinando da essi le aree che hanno subito una variazione sostanziale. Questa tecnica viene utilizzata anche negli algoritmi di compressione video (ad esempio il DivX): si parte da un'immagine di riferimento e delle successive si salvano soltanto le differenze individuate. Per le tecniche di motion detection in real-time, gli algoritmi si differenziano nel modo con cui ricercano il frame di "confronto" detto anche di "background" usato per confrontare ogni immagine successiva, a secondo le specifiche statistico-matematiche scelte. Avrete già capito che la motion detection si basa su un concetto abbastanza semplice:

- Definire un particolare frame, detto background.
- Confrontare tutte le immagini successive con il background.
- Se le sequenze successive differiscono dal background allora c'è stato un movimento nella scena.

Rimane ora da stabilire, come calcolare il background, come ripulire le immagini da eventuali disturbi. Una sequenza semplicemente disturbata, se confrontata con il background potrebbe differire a causa ad esempio di una semplice variazione di luminosità, questo non implica certamente movimento. Bisogna capire cosa è movimento e cosa no. Infine le informazioni sul movimento individuato potrebbero essere collezionate per definire ad esempio una traiettoria o per motivi statistici. Queste varie riflessioni rendono la tecnica della *Motion Detection* molto più interessante rispetto a quanto potrebbe sembrare in un primo momento. Vediamo quali sono gli algoritmi di base per realizzare quanto detto fin qui.

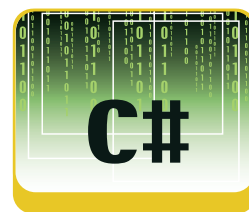
ALGORITMI STATICI

Alcuni di questi algoritmi si basano sul *"metodo di riferimento statico"* nel quale l'immagine di confronto viene scelta staticamente all'inizio del processo esaminando i primi n frame della sequenza; per valutare ogni immagine successiva nella sequenza da ogni pixel di questo frame si determinano dei parametri come il valor medio, la sua varianza e i suoi valori minimi e massimi.
Gli algoritmi a riferimento statico sono in grado di rilevare ogni variazione seppur minima e anche graduale che avviene nella scena di osservazione; il loro impiego è ottimale nel caso in cui lo sfondo di base della scena non si modifichi nel tempo. Se l'immagine di background dovesse subire variazioni non dovuti a un movimento (ad esempio si sposta l'orientamento della videocamera), il frame statico dovrà essere ricalcolato.

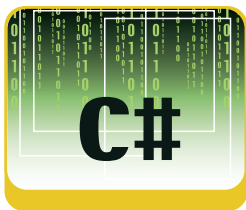
ALGORITMI DINAMICI

In questo caso ci vengono in aiuto i cosiddetti algoritmi adattativi (*Adaptive background*): l'analisi di ogni singola immagine viene effettuata usando i *k frame* ad essa precedenti in modo da ricavare dinamicamente i parametri da usare come immagine di background. In questo caso non ci si basa più su di un'immagine impostata staticamente all'inizio del processo ed i parametri vengono ottenuti assegnandogli cioè un peso tanto maggiore, quanto minore è la distanza temporale dal frame rilevato in tempo reale.
Questi algoritmi ad aggiornamento dinamico dell'immagine di riferimento, tendono a rilevare i cambiamenti occorrenti nella scena in analisi (es. un'auto che entra nell'area visiva della videocamera), ma una volta che il cambiamento diventa persistente esso stesso diventa parte integrante del frame di background (es. l'auto appena entrata nella scena si ferma e rimane in quella posizione). Una volta segnalato un movimento che porta ad un cambiamento in una o più regioni dell'immagine, se tale cambiamento diventa a sua volta stabile, le regioni interessate e modificate diventano a loro volta parte integrante dell'immagine di riferimento. Questa caratteristica fa si che gli algoritmi possano adattarsi meglio anche a cambiamenti naturali, ma graduali (sia globali che locali) della scena, come ad esempio variazioni di luminosità o zone di ombra (es. diversa luminosità durante la giornata oppure passaggio di nuvole in ambienti esterni): questa cosa non è possibile negli algoritmi statici.

ALGORITMI IBRIDI

Esiste una terza categoria di algoritmi ottenuta da un concetto "ibrido" alle prime due *(Weighted update)*, secondo il quale si cerca di bilanciare il calcolo dell'immagine di riferimento utilizzando sia quella acquisita all'inizio del processo, mediata sui primi

FILTRI

Sono particolari algoritmi che aggiungono determinate funzioni di elaborazione dell'immagine nei programmi di grafica. In genere sono composti da una sequenza di istruzioni che modificano l'immagine dando, ad esempio, trasparenza, ondulazione, effetto sferico, ombreggiatura, effetto neon...

I TUOI APPUNTI

n frame (training), sia quella estratta in maniera dinamica sui *k frame* precedenti a quello in esame (dinamica). Si estraggono quindi coppie di parametri relative a tutte e due le immagini di riferimento e successivamente si effettua un bilanciamento di queste coppie di valori in maniera "pesata", definito il parametro *δ∈[0…1]*, i relativi parametri da estrarre per l'immagine di riferimento si ottengono sommando quelli dell'immagine da training moltiplicati per *δ*, con quelli dell'immagine dinamica moltiplicati per *1-δ*. Gli algoritmi basati su questa tecnica sono pensati per includere le caratteristiche sia della prima che della seconda: seppur quindi basandosi su un aggiornamento dinamico dell'immagine di riferimento, tengono anche conto dell'immagine appresa nella fase iniziale. In questo modo si cerca di ovviare alla possibile mancata segnalazione di cambiamenti avvenuti a bassa velocità (che possono non essere rilevati dagli algoritmi adattativi), poiché nonostante tali cambiamenti (oggetti) entrino a far parte del background dell'immagine, si da comunque un certo peso anche al fatto che in fase di apprendimento fossero assenti. Allo stesso modo si cerca di risolvere i problemi relativi ai cambi di luminosità globali della scena, più evidenti nel caso statico.

RED CHANNEL

Tutti i modelli di rappresentazione del colore di un'immagine possono essere visti come "spazi numerici" (tridimensionali per l'RGB e l'HSB e quadrimensionali per il CMYK) cioè delle matrici che indicano l'intensità relativi ai colori per ogni singolo pixel. Estrarre un canale di colore fondamentale vuol dire estrapolare la matrice di quel colore per per l'immagine.

IMAGE PROCESSING

Tutti gli algoritmi di motion detection richiedono sempre un'attenta elaborazione delle immagini della sequenza mediante l'adozione di filtri di compensazione, di correzione e di trasformazione. Ogni frame viene sottoposto a questi filtri per migliorarne la qualità riducendo i "rumori" di base introdotti dalla digitalizzazione della scena e per ottenere i parametri di valutazione dei singoli pixel necessari per confrontarli con l'immagine di background. Abbiamo bisogno quindi di una libreria di *Image Processing* abbastanza potente e completa che ci fornisca gli

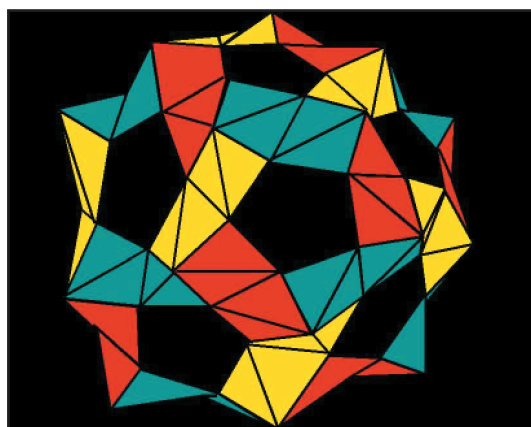



Fig. 2: *L'immagine originale che sottoporremo ai filtri vogliamo applicarvi una sequenza di filtri: una rotazione di 60° e un 'inversione di colori*

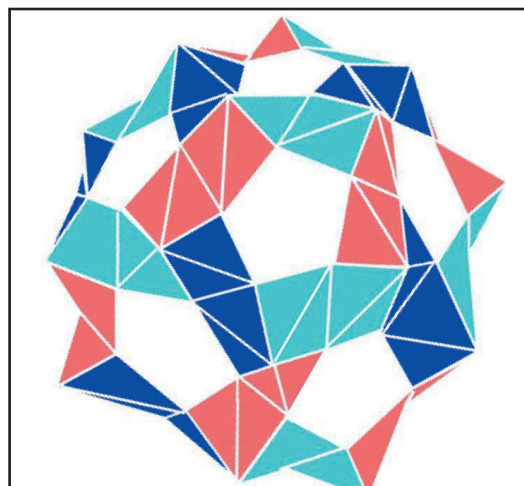



Fig. 3: *Ecco cosa otteniamo una volta sottoposta l'immagine alla sequenza di filtraggio*

strumenti necessari per poter analizzare le immagini di una sequenza video. Ci viene in aiuto la libreria *Tiger* disponibile per la piattaforma Framework .NET, con la quale è possibile sviluppare tutti i filtri di cui abbiamo bisogno e di integrarli facilmente nell'applicazione di analisi che andremo ad implementare. Ad esempio, ad una bitmap letta direttamente da file

```
//carichiamo un'immagine da file
System.Drawing.Bitmap image = (Bitmap)
                    Bitmap.FromFile(fileName);
// definiamo la sequenza di filtri
Tiger.Imaging.Filters.FiltersSequence filter =
        new Tiger.Imaging.Filters.FiltersSequence();
// aggiungiamo i singoli filtri
filter.Add(new Tiger.Imaging.Filters.Rotate(60));
filter.Add(new Tiger.Imaging.Filters.Invert());
// applichiamo i filtri all'immagine originale
System.Drawing.Bitmap newImage =
                    filter.Apply(image);
```

L'analisi di un'immagine tramite operatori locali (che agiscono su un punto traendo informazioni dal suo intorno) consiste in un filtraggio dell'informazione in esso contenuta.

La maggior parte dei filtri implementati nella *Tiger* sono progettati per lavorare con immagini a 24 bit RGB o a toni di grigio: può capitare quindi che un filtro non lavori correttamente dando in uscita un'immagine "svuotata". Per aggirare questo problema è sufficiente sottoporre l'image ad una preelaborazione mediante l'istruzione

```
Tiger.Imaging.Image.FormatImage(ref image);
```

con la quale standardizziamo il formato di lavoro. Da notare che il metodo statico della classe Image richiede come parametro una *System.Drawing.Bitmap* passata come reference: in questo modo trove-

remo il risultato della formattazione nella variabile image stessa. In questo modo l'immagine è stata resa compatibile con tutti i filtri disponibili nella *Tiger.*

IFILTER

Ciascuna trasformazione implementata nella libreria è accomunata alle altre grazie all'interfaccia *IFilter;* una volta scelto l'appropriato filtro da applicare all'immagine, è possibile generalizzarlo lavorando con la sua astrazione *IFilter* senza doverci preoccupare della sua reale implementazione, cioè della classe usata per costruirlo. Applicare un filtro ad una variabile *Bitmap* si riduce a chiamare semplicemente il metodo *Apply* passando come parametro l'image stessa

```
new Tiger.Imaging.Filters.IFilter  filter = new
                          Tiger.Imaging.Filters.Rotate(60);
System.Drawing.Bitmap imageFiltered =
                                       filter.Apply(image);
```

Molte volte può essere necessario sottoporre l'immagine ad una sequenza di trasformazioni (in cascata) prima di passare alle successive fasi dell'elaborazione, come mostrato nell'esempio precedente (una rotazione di 60 gradi e un'inversione di colore): si dovrebbe costruire un filtro per ogni trasformazione richiesta, applicarlo all'immagine ed usare il risultato di questa operazione come parametro al successivo filtraggio. Per rendere atomica l'intera sequenza, cioè racchiuderla in un'unica chiamata al metodo *Apply(),* è possibile costruire un nuovo filtro dalla composizione di altri, naturalmente rispettando l'ordine, usando un *FiltersSequence.*

ALGORITMO DI MOTION DETECTION

Partiamo da un esempio. Un medico deve studiare i movimenti di un paziente addormentato in modo da individuare possibili disturbi del sonno e per fare ciò usa una videocamera puntata sul letto. La maggior parte del tempo quello che verrà ripreso è la persona addormentata sostanzialmente ferma (frame di background). Ad un certo punto il paziente sposta il braccio da sotto il cuscino: confrontando ciascun fotogramma con l'immagine del paziente quando è fermo, si individua per confronto un'area corrispondente al movimento. L'area è composta dai pixel del fotogramma in cui sono state evidenziate delle differenze; per aumentare la leggibilità

Fig. 4: **Lo schema dell'algoritmo di motion detection in grado di evidenziare le aree contenenti figure in movimento.**

MORFOLOGIA MATEMATICA

Molti degli strumenti che abbiamo evidenziato sono disponibili anche in altri lavori simili. Tuttavia per comprendere a fondo cosa abbiamo per le mani, esaminiamo uno dei problemi più ostici del filtraggio delle immagini: la Morfologia Matematica. L'idea sulla quale si fonda la morfologia matematica è, essenzialmente, l'esame della struttura geometrica di un'immagine al fine di rendere evidenti le sue "connessioni topologiche" con un elemento di confronto; tali connessioni dipendono, oltre che dalla geometria della struttura da evidenziare, anche dalla sua posizione all'interno dell'immagine da esaminare. Solo recentemente la morfologia matematica ha acquisito dignità di disciplina autonoma nell'ambito dell'elaborazione delle immagini, il suo impianto matematico si fonda principalmente sulla teoria degli insiemi ed assume in sé concetti di algebra, topologia e geometria. Vediamo come utilizzare la Tiger in tal proposito. Prendiamo una fotografia che riprende il particolare di un palazzo: tramite l'applicazione di questi filtri vogliamo estrarre tutte le linee verticali che compongono il contorno. La prima cosa da fare è esaltare le caratteristiche dell'immagine in modo da estrarre il contorno dell'immagine

```
Tiger.Imaging.Image
        .FormatImage(ref image);
Tiger.Imaging.Filters
 .FiltersSequence sequence=new
          Tiger.Imaging.Filters
            .FiltersSequence();
sequence.Add(new Tiger.Imaging
           .Filters.Pixellate(4));
sequence.Add(new Tiger.Imaging
           .Filters.Threshold());
sequence.Add(new Tiger.Imaging
           .Filters.Dilatation());
sequence.Add(new Tiger.Imaging
              .Filters.Edges());
System.Drawing.Bitmap cImage
        = sequence.Apply(image);
```

Nella Bitmap cImage abbiamo così il contorno dell'immagine. A questo punto non ci resta che costruire l'operatore di filtraggio con il quale isolare dal contorno solamente le linee verticali che lo compongono

```
// matrice di ricerca delle linee
                       verticali
short[,] matrix = new short[3, 3]
{
  {0, 1, 0},
  {0, 1, 0},
  {0, 1, 0}
};
Tiger.Imaging.Filters.HitAndMiss
     verticalFilter = new Tiger
   .Imaging.Filters.HitAndMiss(
                        matrix);
System.Drawing.Bitmap
   verticalImage = verticalFilter
              .Apply(cImage);
```

dell'immagine si evidenzierà soltanto il contorno della zona soggetta al movimento. L'analisi del movimento di oggetti in un video si può ricondurre quindi ad un problema di image processing utilizzando un'opportuna elaborazione dei frame: in sostanza dobbiamo progettare una sequenza di filtri da utilizzare ad ogni passo dell'analisi con cui "manipolare" le singole riprese Nel complesso, l'algoritmo di *motion detection* non presenta difficoltà elevate e non è neppure tanto complicato da implementare proprio grazie alla *Tiger.* Innanzitutto, dopo essere stata sottoposta ad un primo filtro ottenuto dalla composizione del *Grayscale Filter* e del *Pixellate Filter,* ogni nuova scena acquisita concorre a formare il frame di background. Per evidenziale le parti dell'immagine che risultano diverse rispetto al background di riferimento, si predispone una sequenza di filtri che consentirà l'estrazione del contorno degli oggetti in movimento. L'ultimo filtro della sequenza, l'*Edge Filter,* viene utilizzato proprio per individuare i contorni di regioni omogenee per valori di luminosità, ma non basta in quanto in un'immagine ci possono essere confini tra aree aventi una luminosità simile ma che risultano distinte in base ad altre caratteristiche visuali (texture): un classico esempio sono due superfici dello stesso legno affiancate ma con diverse venatura.

L'individuazione dei contorni di un'immagine e la separazione figura/sfondo può essere basata oltre dalla luminosità anche dalle differenze di colore tra le regioni individuabili. Un approccio intuitivo per risolvere l'anomalia di rilevamento è quindi pretrattare l'immagine con l'applicazione di una sequenza di filtri da ognuno dei quali si estrarranno delle informazioni importanti per l'individuazione dei contorni. L'operatore (filtro) *Difference* viene usato per estrarre le differenze assolute tra i corrispondenti simmetrici di tutti i pixel adiacenti, utilizzando come immagine di sovrapposizione il frame di background appena determinato. All'uscita del *Difference* si applica un operatore *Tresholding* il quale rende neri i pixel che hanno un valore inferiore ad una certa soglia (15) e bianchi quelli che superano il massimo impostato (255). A questo punto, prima di applicare l'*Edge Filter* (ed ottenere il contorno del movimento) è necessario trattare l'immagine in modo da "accorpare" in un oggetto unico tutti i pixel simili individuabili, mediante l'operatore *Dilatation* che è un filtro morfologico: ogni singolo pixel assume il valore corrispondente all'oggetto se il numero di pixel adiacenti che appartengono allo stesso oggetto supera un valore si soglia predefiniti (si uniformano i valori). Dall'immagine originale viene poi estratta il *red channel* che andrà composto con il contorno individuato ottenendo il nuovo red channel da settare al posto di quello iniziale.

```
public void ProcessFrame(ref Bitmap image)
{// estrae il red channel dall'immagine originale
 Bitmap redChannel = extrachChannel.Apply(image);
 // Applica il primo filtro alla sequenza
 Bitmap tmpImage = processingFilter1.Apply(image);
 // Se il frame di background non è ancora pronto,
 // sarà la tmpImage appena calcolata
 if (backgroundFrame == null)
 {backgroundFrame = (Bitmap)tmpImage.Clone();
  // non ha senso continuare in quanto le differenze
  // saranno nulle per costruzione
  return;
 }
 //Aggiorna il frame di background ogni due frame
 if (++counter == 2)
 {counter = 0;
  moveTowardsFilter.OverlayImage = tmpImage;
  Bitmap tmp = moveTowardsFilter.Apply(
                                       backgroundFrame);
  backgroundFrame.Dispose();
  backgroundFrame = tmp; }
 // imposta il frame di background come
 //overlay per il Difference Filter
 differenceFilter.OverlayImage = backgroundFrame;
 // Applica il secondo filtro alla tmpImage
 Bitmap tmpImage2 = processingFilter2.Apply(
                                            tmpImage);
 tmpImage.Dispose();
 // Unisce il Red Channel estratto dall'immagine
 // originale con i bordi degli oggetti in movimento
 mergeFilter.OverlayImage = tmpImage2;
 Bitmap tmpImage3 = mergeFilter.Apply(redChannel);
 redChannel.Dispose();
 tmpImage2.Dispose();
 // Sostituisci il red channel dall'immagine originale
 replaceChannel.ChannelImage = tmpImage3;
 Bitmap tmpImage4 = replaceChannel.Apply(image);
 tmpImage3.Dispose();
 image.Dispose();
 image = tmpImage4;
}
```

AGGANCIARSI ALLO STREAM VIDEO

A questo punto non rimane altro da fare che consentire all'utente la scelta della sorgente video da elaborare. Per semplificare un po' si è scelto di limitare l'uso dell'applicazione a due soli casi: video già acquisito o proveniente da una videocamera collegata al computer. Nel primo caso il filmato si troverà nell'hard disk pronto per l'elaborazione *(post-production analyzer),* eventualmente già sottoposto ad una pre-elaborazione in modo da migliorarne la qualità. Il secondo caso è un po' più complesso in quanto bisogna acquisire il video man mano che viene prodotto dalla videocamera e sottoporlo subito all'analisi *(real-time analyzer).*

DIRECTSHOW

DirectShow della Microsoft è un controllo ActiveX , erede di ActiveMovie ed integrato con la tecnologia DirectX, capace di gestire e riprodurre stream (flussi di dati) multimediali di diversa origine (Mpeg, Wav, QuickTime, Avi). Il concetto fondamentale su cui si basa la tecnologia DirectShow è di connettere tra loro diversi filtri ognuno dei quali gestisce una parte del processo di ricezione, decodifica, trasformazione, schedulazione e visualizzazione di flussi video, audio e dati in maniera totalmente interdipendente. Esistono diverse implementazioni (non ufficiali) per Dot.Net

SCELTA DEL DEVICE

Nel caso in cui la scelta del tipo di analisi ricada sul real-time, prima di avviare l'elaborazione l'utente dovrà scegliere quale dispositivo utilizzare tra i dispositivi "fisicamente" connessi al computer.
Quindi il problema da affrontare adesso è quello di recuperare l'elenco dei device attivi (in particolare quelli video) dal sistema e tra questi selezionare quelli dotati della funzionalità di acquisizione video. Non dobbiamo spaventarci all'idea di dover implementare questa attività, in quanto sembra piuttosto complicata ma in realtà è di una semplicità mostruosa.
Per aiutarci nella ricerca dei *Video Input Device,* adoperemo una libreria per Dot.NET la quale fornisce le funzionalità del *DirectShow:* essa mette a disposizione la classe *FilterCollection* che ci consentirà di recuperare i dispositivi di acquisizione audio/video specificando soltanto la funzionalità (o meglio quella che viene anche chiamata la categoria di filtraggio) di cui abbiamo bisogno: *AudioInputDevice* oppure *VideoInputDevice.*
A questo punto, per consentire la scelta del device da parte dell'utente basterà aprire una finestra di dialogo contenente una ComboBox in cui inseriremo l'elenco dei dispositivi trovati

```
dshow.FilterCollection deviceList = new dshow
            .FilterCollection(dshow.Core.FilterCategory
                                   .VideoInputDevice);
  foreach (dshow.Filter device in deviceList)
  {
      deviceCombo.Items.Add(device.Name);
   }
```

VIRTUAL CAMERA

Per l'applicazione non è importante il tipo della sorgente stream nè tanto meno se la sequenza video presentata in ingresso è in presa diretta o registrata: quello che richiede è solamente un flusso di dati, ed è questo quello che le passeremo. Come possiamo rendere l'origine del filmato del tutto trasparente al sistema di elaborazione?
Pensiamo per un momento al funzionamento reale di una videocamera moderna: sul display LCD di cui è dotata possiamo guardare il video durante la ripresa ma anche, una volta riavvolto il nastro, quello che abbiamo appena registrato. L'ideale sarebbe quindi poter utilizzare sempre una videocamera anche per i filmati già realizzati: la soluzione è in realtà un po' forzata perché dovremmo riversare su cassetta le registrazioni per poter procedere all'individuazione dei particolari in movimento durante l'acquisizione attraverso il computer. Però possiamo sempre costruire una videocamera "virtuale" attraverso la quale far passare tutti gli stream, sia quelli relativi ai filmati registrati che quelli in presa diretta, passandoli poi al sistema di elaborazione che li vedrà come prodotti direttamente dalla videocamera virtuale: in questo modo perderemo la "memoria" della loro origine.

IGENERALSOURCE

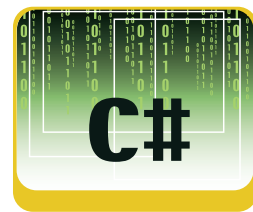
Per generalizzare la sorgente video introduciamo l'interface *IGeneralSource* usata per definire le funzionalità di controllo nell'acquisizione dalle due diverse sorgenti: se l'origine è un file avi useremo la sua classe derivata *FileSource* altrimenti tramite la classe derivata *DeviceSource* eseguiremo il controllo sull'acquisizione in presa diretta. Alla videocamera virtuale, definita attraverso la classe *Camera,* passeremo la generalizzazione della sorgente *IGeneralSource*

```
public interface IGeneralSource
  {
     event CameraEventHandler NewFrameEvent;
     string GeneralSource{get; set;}
     int FramesReceived{get;}
     int BytesReceived{get;}
     object UserData{get; set;}
     bool Running{get;}
     void Start();
     void SignalToStop();
     void WaitForStop();
     void Stop();
   }
```

Per far partire l'acquisizione del video dalla sorgente scelta, agiremo sulla *Camera* la quale chiamerà l'implementazione del metodo *Start()* sull'oggetto generalizzato: sarà quindi compito dell'oggetto recuperare ciascun frame dal flusso e renderlo disponibile alla *Camera.*
Ciascuna entità *DeviceSource* e *FileSource* comunicano la disponibilità di un ogni nuovo frame lanciando un evento *NewFrameEvent(…).*

CONCLUSIONI

Alla luce delle considerazioni fatte e utilizzando alcune librerie facilmente reperibili su internet, non è per nulla complicato realizzare applicazioni che basano la loro principale funzionalità sull'analisi del movimento.
Viceversa il campo di applicazione è decisamente vasto e copre uno spettro enorme tanto da rappresentare una delle nuove frontiere per il futuro della programmazione e del supporto dell'informatica alla qualità della vita.

Ing. Antonino Panella

PER SAPERNE DI PIÙ

Per la realizzazione dell'esempio che mostra una semplice implementazione degli algoritmi di *motion detection* sono stati utilizzati alcuni lavori di Andrew Kirillov sull'elaborazione dell'immagine. Potete trovare alcuni sui interessanti articoli su CodeProject
http://www.codeproject.com/cs/media/Image_Processing_Lab.asp
e http://www.codeproject.com/cs/media/Motion_Detection.asp

remo il risultato della formattazione nella variabile image stessa. In questo modo l'immagine è stata resa compatibile con tutti i filtri disponibili nella *Tiger.*

IFILTER

Ciascuna trasformazione implementata nella libreria è accomunata alle altre grazie all'interfaccia *IFilter;* una volta scelto l'appropriato filtro da applicare all'immagine, è possibile generalizzarlo lavorando con la sua astrazione *IFilter* senza doverci preoccupare della sua reale implementazione, cioè della classe usata per costruirlo. Applicare un filtro ad una variabile *Bitmap* si riduce a chiamare semplicemente il metodo *Apply* passando come parametro l'image stessa

```
new Tiger.Imaging.Filters.IFilter  filter = new
                        Tiger.Imaging.Filters.Rotate(60);
System.Drawing.Bitmap imageFiltered =filter.Apply(image);
```

Molte volte può essere necessario sottoporre l'immagine ad una sequenza di trasformazioni (in cascata) prima di passare alle successive fasi dell'elaborazione, come mostrato nell'esempio precedente (una rotazione di 60 gradi e un'inversione di colore): si dovrebbe costruire un filtro per ogni trasformazione richiesta, applicarlo all'immagine ed usare il risultato di questa operazione come parametro al successivo filtraggio. Per rendere atomica l'intera sequenza, cioè racchiuderla in un'unica chiamata al metodo *Apply(),* è possibile costruire un nuovo filtro dalla composizione di altri, naturalmente rispettandone l'ordine, usando un *FiltersSequence.*

ALGORITMO DI MOTION DETECTION

Partiamo da un esempio. Un medico deve studiare i movimenti di un paziente addormentato in modo da individuare possibili disturbi del sonno e per fare ciò usa una videocamera puntata sul letto. La maggior parte del tempo quello che verrà ripreso è la persona addormentata sostanzialmente ferma (frame di background). Ad un certo punto il paziente sposta il braccio da sotto il cuscino: confrontando ciascun fotogramma con l'immagine del paziente quando è fermo, si individua per confronto un'area corrispondente al movimento. L'area è composta dai pixel del fotogramma in cui sono state evidenziate delle differenze; per aumentare la leggibilità

Fig. 4: ***Lo schema dell'algoritmo di motion detection in grado di evidenziare le aree contenenti figure in movimento.***

MORFOLOGIA MATEMATICA

Molti degli strumenti che abbiamo evidenziato sono disponibili anche in altri lavori simili. Tuttavia per comprendere a fondo cosa abbiamo per le mani, esaminiamo uno dei problemi più ostici del filtraggio delle immagini: la Morfologia Matematica. L'idea sulla quale si fonda la morfologia matematica è, essenzialmente, l'esame della struttura geometrica di un'immagine al fine di rendere evidenti le sue "connessioni topologiche" con un elemento di confronto; tali connessioni dipendono, oltre che dalla geometria della struttura da evidenziare, anche dalla sua posizione all'interno dell'immagine da esaminare. Solo recentemente la morfologia matematica ha acquisito dignità di disciplina autonoma nell'ambito dell'elaborazione delle immagini, il suo impianto matematico si fonda principalmente sulla teoria degli insiemi ed assume in sé concetti di algebra, topologia e geometria. Vediamo come utilizzare la Tiger in tal proposito. Prendiamo una fotografia che riprende il particolare di un palazzo: tramite l'applicazione di questi filtri vogliamo estrarre tutte le linee verticali che compongono il contorno. La prima cosa da fare è esaltare le caratteristiche dell'immagine in modo da estrarre il contorno dell'immagine

```
Tiger.Imaging.Image
        .FormatImage(ref image);
Tiger.Imaging.Filters
 .FiltersSequence sequence=new
            Tiger.Imaging.Filters
              .FiltersSequence();
sequence.Add(new Tiger.Imaging
           .Filters.Pixellate(4));
sequence.Add(new Tiger.Imaging
           .Filters.Threshold());
sequence.Add(new Tiger.Imaging
           .Filters.Dilatation());
sequence.Add(new Tiger.Imaging
               .Filters.Edges());
System.Drawing.Bitmap cImage
          = sequence.Apply(image);
```

Nella Bitmap cImage abbiamo così il contorno dell'immagine. A questo punto non ci resta che costruire l'operatore di filtraggio con il quale isolare dal contorno solamente le linee verticali che lo compongono

```
// matrice di ricerca delle linee
                          verticali
short[,] matrix = new short[3, 3]
{
  {0, 1, 0},
  {0, 1, 0},
  {0, 1, 0}
};
Tiger.Imaging.Filters.HitAndMiss
      verticalFilter = new Tiger
    .Imaging.Filters.HitAndMiss(
                          matrix);
System.Drawing.Bitmap
     verticalImage = verticalFilter
                   .Apply(cImage);
```

PHP & AJAX coppia vincente

Vi presentiamo un metodo che evita il reload delle pagine in seguito alla pressione di un tasto su una form e consente di aggiornare solo le parti destinate a subire variazioni

Questa volta, prima di ogni altra cosa, parleremo un po' di come funziona il web. Per qualcuno potrebbe sembrare un discorso su una tecnologia nota, ed in effetti è proprio così. Tuttavia il modo di gestire il traffico dei dati sul Web è cambiato, anche se il 90% dei programmatori segue ancora i vecchi metodi. Il 90%, ma non tutti, i lettori di ioProgrammo seguendo questo articolo impareranno a usare PHP in congiunzione ad AJAX: un metodo che ottimizza lo scambio dei dati tra il server e il client ed ha come naturale conseguenza quella di produrre web application più veloci e user friendly.

IL METODO TRADIZIONALE

Normalmente un'applicazione per il web che abbia un minimo di interattività utilizza una parte legata all'input, un motore lato server che gestisce le richieste, e infine produce un output. Questa catena è tipicamente costituita da una form html, un linguaggio lato server ad esempio php, e di nuovo una pagina html in output. Facciamo un tipico esempio:

```
<html>
  <head>
    <title>ioProgrammo</title>
    </head>
  <body>
  <img src="logo.jpg"><br><br>
        <form name="formdiselezione" Action=
            "<?$_SERVER["PHP_SELF"]?>" method="Get">
            <select  name="opzioni">
                <option value="1">Fiat</option>
                <option value="2">Ford</option>
                <option value="3">Ferrari</option>
            </select>
        <input type="Submit" name="Invia">
        </form>
  </body>
```

```
</html>
<?
  switch ($_GET['opzioni']) {
    case 1:
      echo "Panda<br>";
      echo "Punto<br>";
      echo "500<br>";
    break;
    case 2:
      echo "Fiesta<br>";
      echo "Focus<br>";
    break;
    case 3:
      echo "Non disponibile<br>";
    break; }
?>
```

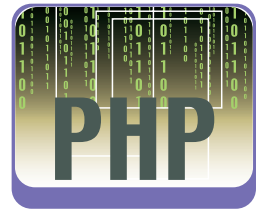
L'applicazione è realmente semplice. L'input è costituito da una casella a tendina, ovvero una select. Quando viene premuto il tasto submit, l'intera pagina viene ricaricata, il contenuto selezionato dall'utente viene passato a PHP che lo analizza e restituisce in output un elenco. È un modo classico di operare in ambiente web. In questo modo di operare ci sono alcuni svantaggi. Prima di tutto si noti che la pagina è composta anche da un'immagine jpg che è posta al di sopra della casellina di selezione. La stessa casellina di selezione è un elemento ripetitivo che compone la pagina. Quando la pagina viene ricaricata viene prodotto un nuovo output che è composto dall'immagine, dalla casella di selezione e da una lista. Tutti gli elementi vengono ricaricati e ridisegnati, anche se in realtà a cambiare è soltanto una piccola porzione della pagina, ovvero quella che contiene la lista. Decisamente uno spreco di risorse.

UNA SOLUZIONE IN JAVASCRIPT

La soluzione che proponiamo di seguito, fa uso di

una funzione javascript e di un tag *div*. Molto semplicemente eviteremo il reload della pagina passando alla funzione javascript il contenuto selezionato dall'utente e usando la propietà *innerHTML* del campo *div* per scrivere dinamicamente sulla pagina. Vediamo come:

```
<html>
  <head>
     <title>ioProgrammo</title>
     </head>
  <body>
  <script type="text/javascript">
  function selector(innerlist) {
  var innervalue= innerlist.options
                          [innerlist.selectedIndex].value
     if (innervalue==1) {
     document.getElementById('lista').innerHTML =
                              "Panda<br>Punto<br>500";
     }
     if (innervalue==2) {
     document.getElementById('lista').innerHTML =
                                     "Fiesta<br>Focus";
     }
      if (innervalue==3) {
     document.getElementById('lista').innerHTML =
                                    "Non disponibile";
     }
  }
  </script>
  <img src="logo.jpg"><br><br>
            <form name="formdiselezione"
                                         method="Get">
            <select onchange="selector(this)"
                                           id="opzioni">
             <option value="1">Fiat</option>
             <option value="2">Ford</option>
             <option value="3">Ferrari</option>
            </select>
         </form>
     <div id="lista">
     </div>
  </body>
</html>
<?
?>
```

Il codice che vi abbiamo proposto , migliora di molto l'efficienza dello script, perché evita completamente il reload della pagina. D'altra parte presenta uno svantaggio enorme. Nel caso semplice, ovvero quello che abbiamo analizzato tutto sembra funzionare nel migliore dei modi. Consideriamo però un caso che faccia uso di un database, ad esempio il database di tutti i comuni di italia. Nella casella a scorrimento potrebbero esserci tutte le lettere dell'alfabeto, selezionando una lettera vorremo avere come risultato quello di ottenere tutti i comuni il cui nome inizia con la lettera selezionata. Per poter ottenere questo risultato, in fase di loading della pagina dovremmo generare una funzione javascript che precarichi i comuni dell'intero database per poter poi fare apparire solo quelli desiderati tramite la proprietà *innerHTML* che abbiamo già usato. Da un punto di vista strettamente programmatico questo non comporta alcuna difficoltà, sarebbe sufficiente un ciclo di *while* che scorre gli elementi del database e genera la funzione javascript. D'altra parte questo significa avere all'interno della stessa pagina l'intero elenco di tutti i comuni cosa che l'appesantisce notevolmente. Di fatto la maggior parte delle informazioni contenute nella pagina non ci serviranno mai, il risultato sarà quello di velocizzare la visualizzazione della lista, ma di ottenere un tempo di caricamento della pagina decisamente alto, perciò il nostro problema è tutt'altro che risolto.

LA SOLUZIONE AJAX

Ajax risolve completamente il nosto problema. Di fatto fa uso di un meccanismo chiamato *"XMLHttpRequest"*, che consente di "servire" con una request soltanto una porzione di pagina. Tanto per schematizzare in modo semplice il procedimento, prima di passare ad un esempio pratico:

1) Viene prodotta una pagina html contenente la sola casella con l'elenco delle lettere dell'alfabeto.
2) In relazione all'evento *OnChange* sulla casella di selezione viene inviata una richiesta ad un file .php.
3) La richiesta viene fatta utilizzando *XMLHttpRequest*, perciò la pagina non viene ricaricata, viceversa l'output prodotto dalla pagina php che elabora i dati alimenterà la lista dei comuni presente nella pagina base.
4) La lista dei comuni verrà aggiornata tramite Javascript utilizzando la proprietà *innerHTML*.

Il risultato è evidente, ovvero nessun reload della pagina, nessun precaricamento di dati, scambio di messaggi solo in relazione ai dati richiesti.
Vediamo come tutto questo è possibile.
Avremo bisogno di tre file:

1) Il file che definisce la form che ci servirà per inserire i dati;
2) un file javascript che conterrà le funzioni che ci serviranno per gestire l'output;
3) un file php che elaborerà le richieste e restituirà un elenco corretto.

Il file che definisce la form si chiamerà *index.php* e conterrà il seguente codice:

```
<html>
  <head>
    <title>ioProgrammo</title>
    <script language="javascript" type=
                        "text/javascript" src="./iop.js">
    </script>
  </head>
  <body>
  <div id="caselladiselezione"
  <img src="logo.jpg"><br><br>
        <form id="formdiselezione" method="Get">
        <select onchange="selector()" id="opzioni">
         <option value="1">Fiat</option>
         <option value="2">Ford</option>
         <option value="3">Ferrari</option>
        </select>
      </form>
  </div>
    <div id="lista">
    </div>
  </body>
</html>
```

Le uniche note da mettere in evidenza rispetto a questo codice sono relative all'inclusione del file javascript *iop.js* e alla funzione *selector()* che verrà scatenata dall'evento *OnChange* sulla casella di selezione e che definiremo appunto in *iop.js*.
Il file *iop.js* conterrà il seguente codice:

```
function CreaOggetto(){
  var richiesta;
  var browser = navigator.appName;
  if(browser == "Microsoft Internet Explorer"){
    richiesta = new ActiveXObject(
                          "Microsoft.XMLHTTP");
  }else{
    richiesta = new XMLHttpRequest();
  }
  return richiesta;
}
var http = CreaOggetto();
function selector(){
  http.open('get', 'process.php?lettera='+ document
           .formdiselezione.opzioni.options[ document
         .formdiselezione.opzioni.selectedIndex].value);
  http.onreadystatechange = gestisciContenuto;
  http.send(null);
}
function gestisciContenuto(){
  if(http.readyState == 4){
     var response = http.responseText;
     document.getElementById('lista').innerHTML =
                                          response;
                                                 }
}
```

La prima function *CreaOggetto()*, controlla qual è il browser che sta cercando di utilizzare un *XMLHttprequest*, infatti l'oggetto deve essere creato in modo diverso per explorer o per mozilla.
La function *Selector()* è quella che viene scatenata dall'evento *OnChange* sulla casella a discesa. Il suo scopo è evidente, costruisce semplicemente una richiesta get con un parametro da passare all'oggetto http di classe *XMLHttpRequest()*, sarà questo oggetto il responsabile della comunicazione con il file *process.php*. *XMLHttpRequest* azionerà una sorta di "tunnel" invisibile all'utente, tramite questo tunnel la richiesta della pagina *index.php* verrà processata dal file *process.php* nonostante questo nessuna pagina verrà ricaricata, tutto avverrà con uno scambio di messaggi fra client e server che non richiede però che la pagina venga ricaricata. Il contenuto dell'elaborazione di *process.php* sarà contenuto in una variabile response, dichiarata nella function *gestisciContenuto*. L'unica nota da fare rispetto a questa function è relativa alla proprietà http *.readyState*. Di fatto in ogni momento lo stato della richiesta può essere uno dei seguenti:

```
0: Uninitialized
1: Loading
2: Loaded
3: Interactive
4: Finished */
```

Lo stato che interessa a noi è ovviamente il 4.
Nessun valore può essere visualizzato fino a che la richiesta non è stata processata infatti.
Per concludere la lista viene aggiornata tramite la propery *innerHTML*.
Non ci resta che visualizzare il contenuto di *process.php*.

```
<?
 switch ($_GET['lettera']) {
   case 1:
     echo "Panda<br>";
     echo "Punto<br>";
     echo "500<br>";
   break;
   case 2:
     echo "Fiesta<br>";
     echo "Focus<br>";
   break;
   case 3:
     echo "Non disponibile<br>";
   break;
   default: echo "";
 }
?>
```

In questo caso, molto banalmente ci siamo limitati ad uno switch, l'idea era quella di mostrare la tecnica, ovviamente in casi di produzione molto proba-

bilmente in questo script saranno contenute le istruzioni per recuperare i dati da un database.

UNA RISPOSTA IN XML

Potrebbe essere interessante applicare quanto fin qui detto a risposte di tipo XML anziché di puro testo. Cambiamo leggermente il file process.php, come segue:

```
<?php
header('Content-Type: text/xml');
echo '<?xml version="1.0" encoding="UTF-8"
                                    standalone="yes"?>';
function valorizza($p) {
  switch ($p) {
     case 1:
        return array("Panda","Punto","500");
     break;
     case 2:
        return array("fiesta");
     break;
     case 3:
        return array("non disponibile");
     break;
     default: return ""; }
}
?>
<response>
  <?
  foreach (valorizza($_GET['lettera'])  as $valore ) {
  ?>
  <mezzo><?echo $valore?></mezzo>
  <? }
  ?>
</response>
     case 3:
        echo "<response>";
        echo "<mezzo>Non disponibile</mezzo>";
        echo "<description>Il sogno nascosto</mezzo>";
        echo "</response>";
     break;
     default: echo "";}
?>
```

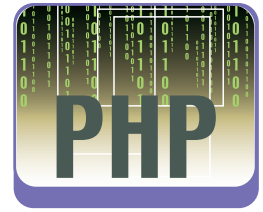
e di conseguenza il file iop.js

```
function CreaOggetto(){
        var richiesta;
        var browser = navigator.appName;
        if(browser == "Microsoft Internet Explorer"){
                richiesta = new ActiveXObject(
                                    "Microsoft.XMLHTTP");
        }else{
                richiesta = new XMLHttpRequest(); }
        return richiesta; }
var req = CreaOggetto();
```

```
function selector(){
   var url =  'process1.php?lettera='+
                document.formdiselezione.opzioni.options
     [document.formdiselezione.opzioni.selectedIndex]
                                                .value;
   req.open('get',url,true);
   req.onreadystatechange = gestisciContenuto;
   req.send(null);}
function gestisciContenuto(){
   if(req.readyState == 4){
      var risposta = req.responseXML.documentElement;
      var mialista = ""
      mezzi = risposta.getElementsByTagName('mezzo');
      for(var i=0; i < mezzi.length; i++){
          mialista = mialista+mezzi[i].firstChild.data+
                                                 '<br>';}
      document.getElementById('lista').innerHTML =
                                            mialista; }
}
```

In process1.php abbiamo cambiato la struttura di modo che il file generato sia in formato xml. Da notare l'istruzione *header('Content-Type: text /xml');* senza questa ci potrebbero essere problemi di parsing. Nel file *iop.js* abbiamo modificato *var risposta = req.responseXML.documentElement;* che adesso sfrutta un metodo per caricare l'XML invece del precedente *var response = http.responseText;* e di conseguenza viene efettuato un parsing del file xml. Per quest'ultima parte ci potrebbero essere altri modi siadi generare il file xml sia di parserizzarlo, al momento non è scopo di questo articolo, mostrare le tecniche di parsing di un file xml. L'importante invece è comprendere come AJAX possa aiutarci nel manipolare anche questo formato di file.

CONCLUSIONI

Ajax è uno strumento interessante che migliora ulteriormente lo scambio dati fra client e server. In questo articolo ne abbiamo mostrato un'integrazione con PHP, ma in realtà il file che processa le richieste può essere generato in un qualunque linguaggio, Ajax è un oggetto che viene instanziato dal browser, un ActiveX nel caso di Internet Explorer, un oggetto nativo nel caso di Mozilla, questo lo rende completamente indipendente dal linguaggio con cui le richieste vengono processate lato server. C'è anche da aggiungere che se da un lato viene ottimizzato lo scambio dei dati, dall'altro viene leggermente appesantita la pagina dalla necessaria presenza degli script JavaScript e data la necessità di instanziare l'oggetto *XMLHttpRequest*, a fronte di quest'aumento della complessità tuttavia i vantaggi in termini di prestazioni ci sembrano innegabili. Indubbiamente si tratta di una tecnica che ci aiuterà molto nel corso dello sviluppo dei nostri progetti.

Quando PHP Incontra J2ME

In questo articolo impareremo a far comunicare Java da un dispositivo mobile, con un server sul quale gira PHP. Scopriremo gli innumerevoli vantaggi forniti da questo tipo di approccio

J2ME *(Java 2 Platform Micro Edition)* è la versione di Java dedicata ai dispositivi mobili. MIDP può essere considerato l'elemento chiave di J2ME ed è la parte orientata ai telefoni cellulari. PHP è, senza dubbio, uno dei linguaggi di programmazione più utilizzati per lo sviluppo di Web application. La potenza di PHP è dovuta, tra le altre cose, al fatto che sia open source. Vi è in rete una vasta community dedicata a questo potente linguaggio e questo ne fa crescere il suo sviluppo in modo esponenziale.
In questo articolo vedremo come questi due potenti linguaggi possono essere utilizzati in accoppiata per sviluppare applicazioni client/server veramente potenti. Svilupperemo, a tal proposito, un'applicazione che ci permetterà di calcolare il codice fiscale di una persona, direttamente dal telefono cellulare, inserendo i suoi dati anagrafici. La novità non sta ovviamente nell'algoritmo per il calcolo del codice fiscale, quanto all'integrazione di un'applicazione J2ME con uno script lato server scritto in PHP.

COMUNICAZIONE TRA J2ME E PHP

Come spesso avviene nel "nostro" mondo, ovvero quello dell'informatica, quando due entità non parlano la stessa "lingua" allora viene utilizzato un protocollo comune che ci permetta comunque di farle comunicare. Nel nostro caso l'artificio utilizzato è quello di far "parlare" J2ME e PHP utilizzando HTTP. L'*Hyper Text Transfer Protocol* (HTTP) è, senza alcun dubbio, uno dei protocolli principalmente utilizzati nella rete mondiale. Ovviamente, lo scopo di questo articolo non è quello di entrare nei meandri dell'HTTP, né tanto meno è richiesto esserne esperti per poter procedere nella lettura. In questo contesto, è sufficiente sapere che HTTP è un protocollo di tipo request/response. Questo significa, semplicemente, che il client fa una richiesta *(request)* al server e quest'ultimo risponde *(response)* al client in modo opportuno. Naturalmente, questa richiesta e risposta dovranno essere fatte rispettando il protocollo capito da entrambi i linguaggi, cioè HTTP.
Per chiarire il concetto facciamo un esempio.
Se in PHP scriviamo qualcosa del genere:

```
echo "ioProgrammo";
```

il risultato dipenderà dal client che sta effettuando la richiesta. Se il client è un classico Web browser, come Internet Explorer o Mozilla Firefox, allora il risultato di quel codice sarà la visualizzazione della stringa ioProgrammo all'interno del corpo del browser. Se invece il client è, come nel nostro caso, un dispositivo mobile, allora quest'ultimo riceverà la stringa ioProgrammo come uno stream di byte. Chiaramente, sarà compito del dispositivo interpretare questi byte in modo opportuno.

COME INIZIARE

Sul proprio computer è necessario aver installato il wireless toolkit e un compilatore java. Sul server è necessario avere installato un Web Server completo di PHP.
I passi per iniziare sono semplici, è sufficiente utilizzare il wizard del Wireless Toolkit come segue:

- **Avviamo il Wireless Toolkit**
- **Selezioniamo New Project**
- **Selezioniamo il nome del progetto e della MIDlet attribuendogli i nomi CFWIreless**
- **Clicchiamo su Create Project**
- **Clicchiamo su OK quando viene aperta la finestra delle impostazioni.**

Non resta che scrivere il codice come descritto nell'articolo.

Tutto ciò è possibile grazie al fatto che sia il client sia il server utilizzano lo stesso protocollo, ovvero HTTP.

IL NOSTRO PROGETTO

Quanti di voi hanno avuto la necessità, almeno una volta, di conoscere il codice fiscale (da ora in poi CF) di una persona, magari la madre o il fratello, e per farlo l'hanno dovuta contattare telefonicamente? Credo molti. A me personalmente è capitato molto più di una volta! Per tale motivo l'applicazione che svilupperemo ci permetterà di calcolare un CF, a partire dai dati anagrafici della persona, avendo a disposizione un dispositivo Java-enabled, come un cellulare di ultima generazione. Intendiamoci, sviluppare un programma per il calcolo del CF non è nulla di così complicato se si ha a disposizione un PC. Per tale motivo non entreremo nel dettaglio dell'implementazione lato server che dovrà fare proprio questo. Approfondiremo, invece, lo sviluppo lato client e l'interazione client/server.

Fig. 1: *Interfaccia grafica della nostra applicazione*

LATO CLIENT

Per ragioni di spazio non verrà analizzato tutto il codice relativo al progetto. Potete comunque trovare il codice completo e commentato nel CD allegato alla rivista. Il nostro lato client sarà costituito da un cellulare in grado di supportare un'applicazione scritta in J2ME. Il codice che scriveremo avrà la responsabilità di:

- Costruire una GUI che permetta all'utente di inserire i dati anagrafici della persona e richiedere il calcolo del corrispondente CF.
- Inviare i dati anagrafici allo script PHP lato server che si occuperà di utilizzarli per il calcolo effettivo del CF.
- Ricevere il response dal server e visualizzarlo all'utente.

Tutto ciò sarà implementato utilizzando tre classi:

- **CFWireless**, responsabile per la GUI.
- **CFComputer**, che si occuperà della comunicazione col server.
- **Person**, la quale rappresenterà, per il nostro sistema, una persona attraverso i suoi dati anagrafici.

L'interfaccia grafica sviluppata permette di inserire i seguenti dati: cognome, nome, sesso, data di nascita e comune di nascita. È infatti sulla base di queste informazioni che verrà calcolato il CF. Il seguente codice, estratto da CFWireless, contiene tutti gli elementi della nostra GUI:

```
private Form fmMain;
```

I TUOI APPUNTI

PERCHÉ PHP?

A mio avviso, capire il motivo per il quale viene seguito un approccio al posto di un altro è la chiave per poterne apprezzare tutti i vantaggi. Alcuni di voi avranno già letto qualche articolo in cui viene mostrato come far interagire J2ME, da un dispositivo mobile, con una servlet che gira su un Web server. Ovviamente, dato che entrambi parlano la stessa "lingua", cioè Java, questo è l'approccio che comunemente viene seguito nello sviluppare applicazioni client/server di questo tipo. Il problema principale di questa soluzione, è che la maggior parte delle società che offrono un servizio di hosting ad un prezzo piuttosto basso supportano PHP ma non Java, quindi non è possibile caricare la servlet sviluppata sul proprio dominio per poterla "realmente" utilizzare. Questo è il primo, non irrilevante, motivo per il quale è utile apprendere come far comunicare PHP e J2ME. Un secondo vantaggio è costituito dal fatto che PHP è un linguaggio ampiamente utilizzato e, come già detto, gode di una community open source abbastanza vasta. Questo fa sì che la rete sia piena di script PHP già "preconfezionati" che rispondono ai nostri scopi e liberamente utilizzabili. È possibile, infatti, trovare script di matematica, fisica, per il parsing di feed RSS e chi più ne ha più ne metta!

```
//cognome
private TextField tfLast;
//nome
private TextField tfFirst;
//comune
private TextField tfPlace;
//sesso
private ChoiceGroup cgGender;
//data nascita
private DateField dfDate;
//comandi
private Command cmExit;
private Command cmNew;
private Command cmCompute;
```

Come è possibile vedere c'è un form, *fmMain*, che contiene tutti gli elementi visti, inclusi i tre bottoni (comandi) seguenti: *cmExit, cmNew, cmCompute*. Essi servono, rispettivamente, per uscire dall'applicazione, resettare i campi in modo da calcolare un nuovo CF, e computare un CF con i dati presenti nei campi. Il costruttore non fa altro che istanziare opportunamente i campi appena visti ed aggiungerli al form principale, ovvero *fmMain*. Quest'ultimo verrà poi impostato come display corrente nel metodo *startApp* invocato automaticamente al lancio dell'applicazione. È più interessante analizzare in dettaglio, invece, il metodo *commandAction* invocato ogni volta che viene selezionato un comando.
Il codice relativo ad esso è il seguente:

```
public void commandAction(Command c, Displayable d) {
  //comando Esci
  if(c == cmExit)
    destroyApp(false);
    //comando Nuovo -> resetta i campi
    else if(c == cmNew) {
      tfLast.setString("");
      tfFirst.setString("");
      tfPlace.setString("");
      dfDate.setDate(null);
    }
    //comando Calcola -> invia i dati al server per
                                        calcolare il CF
   else if(c == cmCompute) {
      if(tfLast.getString() == "" || tfFirst.getString()
                  == "" || dfDate.getDate() == null
                        ||tfPlace.getString() == "") {
        showAlert("Tutti i campi sono obbligatori",
                                            fmMain);
      return;
    }
    Person person = new Person(tfLast.getString(),
          tfFirst.getString(), cgGender.getString(
      cgGender.getSelectedIndex()), dfDate.getDate(
                ).toString(), tfPlace.getString());
    //crea un'istanza della classe che comunica con il
                                              server
      CFComputer cfc = new CFComputer(person, this);
      //mostra un alert indicante che il calcolo del CF è
                                            in corso
      showAlert("Calcolo CF in corso...", fmMain);
      //inizia il calcolo vero e proprio
      cfc.start();
  }
}
```

Questo metodo controlla quale comando è stato selezionato. Se è *cmExit*, l'applicazione viene terminata. Se la scelta è *cmNew* allora vengono resettati tutti i campi per permettere il calcolo di un nuovo CF. Se il comando selezionato è, invece, *cmCompute*, allora per prima cosa viene effettuato un controllo sul riempimento di tutti i campi, nel caso in cui qualche campo non sia stato compilato visualizzato un *alert* per notificare l'errore. Dopodiché viene instanziato un oggetto di tipo *Person* e passato come parametro, insieme alla *midlet* corrente, al costruttore della classe *CFComputer* che vedremo tra poco. Essa è responsabile della comunicazione col server. La classe *Person* si occupa di mettere i campi, ricevuti attraverso il suo costruttore, in una forma opportuna. Per chiarire, diamo un'occhiata all'inizio della classe *Person:*

```
class Person {
  private String lastName;
  private String firstName;
  private String gender;
  private String dayBirth;
  private String monthBirth;
  private String yearBirth;
  private String placeBirth;
  public Person(String lastName, String firstName,
    String gender, String dateBirth, String placeBirth) {
    this.lastName = fixData(lastName);
    this.firstName = fixData(firstName);
    this.gender = gender;
    this.dayBirth = getDay(dateBirth);
    this.monthBirth = getMonth(dateBirth);
    this.yearBirth = getYear(dateBirth);
    this.placeBirth = fixData(placeBirth);
}
...
```

Come si può vedere, nel costruttore viene chiamato il metodo *fixData* sui campi relativi a cognome *(lastName)*, nome *(firstName)* e comune di nascita *(placeBirth)*. Il metodo *fixData* è così definito:

```
private String fixData(String data) {
  //sostituisce le vocali accentate
  data = fixVowels(data);
```

```
    //rimuove gli spazi in eccesso
    data = trimSpaces(data);
    //rende tutto maiuscolo
    data = data.toUpperCase();
    return data;
}
```

Come si può notare dai commenti il metodo è "autodescrittivo" e non richiede ulteriori chiarimenti. I metodi *getDay*, *getMonth* e *getYear* sono usati, rispettivamente, per estrarre giorno, mese e anno di nascita dalla stringa rappresentante la data di nascita della persona. Dopo la creazione di un'istanza della classe *Person* viene visualizzato un *alert* indicante che il calcolo del CF è in corso e viene invocato il metodo *start* della classe *CFComputer* per il calcolo vero e proprio.
La classe *CFComputer* inizia nel seguente modo:

```
class CFComputer implements Runnable {
    private Person person;
    private String URL="http://localhost:8000/CF/cf.php";
    private CFWireless midlet;
    private String CF;
    public CFComputer(Person person, CFWireless
                                                midlet) {
        this.person = person;
        this.midlet = midlet;
    }
    public void run() {
        try {
            getCF();
        }
        catch(Exception e) {
            midlet.showCF("");
        }
    }
    public void start() {
        Thread t = new Thread(this);
         try {
             t.start();
        }
        catch(Exception e) {
            midlet.showCF("");
        }
    }
    private void getCF() throws IOException {
        InputStream is = null;
        StringBuffer sb = new StringBuffer();
        HttpConnection http = null;
        //costruisce la query string
        URL += "?l="+person.getLastName()+
        "&f="+person.getFirstName()+
        "&g="+person.getGender()+
        "&d="+person.getDayBirth()+
        "&m="+person.getMonthBirth()+
        "&y="+person.getYearBirth()+
        "&p="+person.getPlaceBirth();
        //sostituisce i caratteri non permessi in un URL
        URL = encodeURL(URL);
        try {
            //stabilisce la connessione
            http = (HttpConnection)Connector.open(URL);
            //imposta GET come metodo di request
              http.setRequestMethod(HttpConnection.GET);
            //riceve la response dal server
            if(http.getResponseCode() ==
                                HttpConnection.HTTP_OK) {
            int ch;
            is = http.openInputStream();
            while((ch = is.read()) != -1)
            sb.append((char)ch);
          }
      } catch(Exception e) {
        midlet.showCF("");
      } finally {
        if(is != null)
            is.close();
            if(sb != null)
              CF = new String(sb);
            else
              CF = new String();
              if(http != null)
                http.close();
              }
              midlet.showCF(CF);
        }
...
```

Tale classe implementa l'interfaccia *Runnable* in modo da effettuare la comunicazione con il server in un *thread* separato. Una cosa da chiarire riguarda l'URL seguente:

```
http://localhost:8000/CF/cf.php
```

Esso ha questa forma in quanto, sulla mia macchina, ho impostato il Web Server Apache in ascolto sulla porta 8000 poiché la porta di default, ovvero l'80, era già utilizzata da IIS.
Chiaramente dovrete adattare l'URL alle vostre impostazioni. Inoltre, quando caricherete i file *.php* su un Web server "vero", dovrete cambiare opportunamente l'URL di riferimento in una forma simile alla seguente:

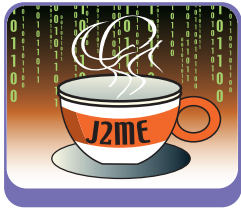
```
http://www.iltuosito.it/CF/cf.php
```

È possibile notare che il metodo che effettua il lavoro vero e proprio è *getCF*. In questo metodo viene costruita la query string accodandola all'URL di riferimento. Gli elementi della query string sono ricavati dall'oggetto di tipo *Person* visto precedentemente. L'URL viene, poi, opportunamente modificato rimpiazzando i caratteri non permessi, come lo spazio che viene sostitui-

J2ME Wireless Toolkit
http://java.sun.com/products/j2mewtoolkit

PHP
http://www.php.net

Apache Web Server
http://www.apache.org

J2ME white papers
http://java.sun.com/j2me/reference/whitepapers

to da %20. A questo punto viene la parte un po' più delicata che può essere riassunta in tre passi:

- Si crea una connessione di tipo *HttpConnection*.
- Si imposta GET come metodo di richiesta.
- Si riceve la risposta dal server e, in caso questa sia *"OK"*, viene costruito uno *StringBuffer* con lo stream di byte ricevuto dal server.

Se durante tutto ciò si verifica un'eccezione viene chiamato il metodo *showCF*, della classe *CFWireless*, passandogli una stringa vuota. Se non avviene l'eccezione, invece, gli viene passata la stringa ricevuta dal server che, se tutto è andato liscio, è proprio il CF della persona.
Diciamo "se tutto è andato liscio" poiché se, ad esempio, il comune non è presente in archivio la stringa ritornata dal server sarà "Comune non presente" e non il CF atteso. Il codice di *showCF* è il seguente:

```
public void showCF(String CF)
{
  if(CF == "") //calcolo fallito
    showAlert("Calcolo fallito per problemi di rete.
                    Riprova più tardi", fmMain);
  else  //calcolo OK
  showAlert("Codice Fiscale: "+CF, fmMain);
}
```

Fig. 2: *L'applicazione in fase di esecuzione*

Esso mostra un *alert* a seconda della stringa ricevuta. Se è una stringa vuota il calcolo è fallito, altrimenti o viene visualizzato il CF o la stringa ricevuta, ad esempio "Comune non presente". La **Figura 2** mostra il risultato del calcolo del CF di una persona con i seguenti dati:

- **Cognome**: ROSSI
- **Nome:** MARIO
- **Sesso:** M
- **Data di nascita:** 01/01/1976
- **Comune di nascita:** COSENZA

Non so se realmente esista una persona con questi dati. In caso affermativo, chiediamo venia per questa "invasione della privacy".

LATO SERVER

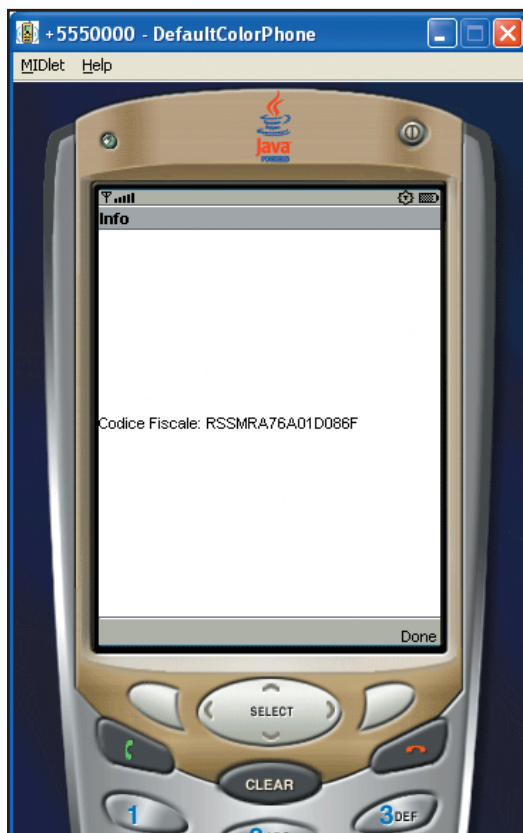
Come già detto, non analizzeremo il lato server in dettaglio dato che si tratta di una cosa abbastanza semplice dal punto di vista della programmazione. Descriveremo solo l'algoritmo per il calcolo del CF e qualche passaggio chiave del codice ad esso dedicato. Il CF di una persona è costituito da una stringa di undici caratteri.
Tornando all'esempio precedente abbiamo:

RSS MRA 76A01 D086 F

Ho utilizzato gli spazi per suddividere la stringa in cinque parti per evidenziare meglio le singole porzioni. I primi tre caratteri vengono ricavati dal cognome di una persona. I successivi tre dal nome. Poi vi sono cinque caratteri identificanti la data di nascita ed il sesso. Il comune è indicato dai quattro caratteri che seguono. L'ultima lettera è ricavata tramite un calcolo particolare dipendente dai dieci caratteri precedenti. L'algoritmo è il seguente. I caratteri del cognome vengono fuori dalle prime tre consonanti di esso. In caso il cognome abbia solo due consonanti viene presa la prima vocale.
Ad esempio *NERI* sarà *NRE*. Se la consonante presente è una sola vengono prese le prime due vocali. In caso il cognome sia, invece, costituito da due caratteri soltanto, verrà completato con una *X*. Ad esempio CARLO BO avrà *BOX* come prime tre lettere del suo CF. Ritengo abbastanza difficile l'esistenza di una persona, in Italia, con un cognome costituito da una sola lettera! Comunque in quel caso le *X* aggiunte saranno due invece di una. Le tre lettere dedicate al nome seguono una procedura simile. L'unica differenza è che, se sono disponibili almeno quattro consonanti, vengono selezionate la prima la terza e la quarta.

NOTE

MIDP

MIDP *(Mobile Information Device Profile)*, assieme a CLDC *(Connected Limited Device Configuration)*, fornisce l'ambiente di runtime per tutti i moderni telefoni cellulari. Infatti, le specifiche MIDP furono definite, attraverso la JCP *(Java Community Process)*, tramite collaborazione di oltre 50 compagnie legate al mondo wireless. Ovviamente, anche in J2ME, continua a valere la filosofia Java, ovvero *"Write once, run everywhere"*.

Ad esempio, il mio nome completo è ANTONINO ALESSANDRO LACAVA. Dato che ANTONINO è il mio primo nome e dato che è formato da quattro consonanti, le tre lettere saranno NNN, ovvero prima, terza e quarta. Negli altri casi verrà seguita la stessa procedura vista per il cognome. Dei cinque caratteri della data i primi due costituiscono l'anno, il terzo il mese e gli ultimi due il giorno. Come detto, da questa stringa, è possibile identificare anche il sesso. Infatti, in caso di sesso femminile, al giorno di nascita viene aggiunto 40 per identificare in modo univoco una donna.

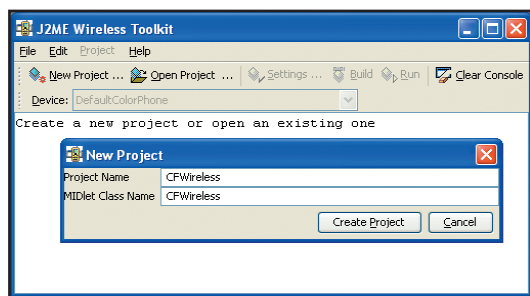



Fig. 3: ***sss***

Ad esempio, nel caso di persona di sesso *F* nata il 02/04/1982 il codice sarà *82D42*. Le quattro lettere seguenti rappresentano il codice assegnato al comune. L'ultima lettera, invece, viene ricavata tramite un algoritmo che sostituisce i dieci caratteri precedenti con delle cifre. Queste vengono, poi, sommate ed il risultato diviso per 26. Il resto di questa divisione viene convertito in una lettera dell'alfabeto.

Come avete avuto modo di notare dall'URL già visto, lo script a cui vengono inviati i dati è *cf.php* il cui codice è:

```
<?php
require("calcolo.inc.php");
if(isset($_GET["l"]) && isset($_GET["f"]) &&
                                  isset($_GET["d"]) &&
   isset($_GET["m"]) && isset($_GET["y"]) &&
                                  isset($_GET["g"]) &&
   isset($_GET["p"])) {
      $monthBirth = $_GET["m"];
      $lastName = $_GET["l"];
      $firstName = $_GET["f"];
      $dayBirth = $_GET["d"];
      $yearBirth = $_GET["y"];
      $gender = $_GET["g"];
     $placeBirth = $_GET["p"];
     $codFisc = calcolaCodiceFiscale($lastName,
                    $firstName, $dayBirth, $monthBirth,
                      $yearBirth, $gender, $placeBirth);
      echo $codFisc;

}
```

```
else {
  echo "Errore nella query string";
}
?>
```

Questo script si occupa solo di ricevere i dati dal client e passarli alla funzione *calcolaCodiceFiscale* che si trova nel file *calcolo.inc.php* assieme ad altre funzioni di supporto. Per ragioni di spazio non vedremo queste funzioni che, come è stato possibile notare dall'algoritmo per il calcolo del CF, non sono nulla di complicato.

CONCLUSIONI

Vorrei spendere solo due parole sulla memorizzazione del codice dei comuni. Si è scelto, a tal fine, di utilizzare file di testo in modo da non costringere l'applicazione ad appoggiarsi su un database. Quest'ultima sarebbe stata, infatti, una scelta migliore ma ci avrebbe costretto ad avere un DB per realizzare "realmente" la nostra applicazione. Inoltre sono stati creati tanti file di testo quante sono le lettere dell'alfabeto. Ad esempio, i comuni che iniziano per *"A"* sono stati memorizzati nel file *comuniA.txt*, quelli per *"B"* in *comuniB.txt* e così via. Questo è stato fatto per ragioni legate alle performance in quanto la ricerca viene così velocizzata. Ovviamente avremmo potuto memorizzare i comuni in ordine alfabetico ed utilizzare algoritmi di ricerca particolari, come la ricerca binaria, ma ciò avrebbe spostato il focus dell'articolo su altre argomentazioni. Lo scopo era, invece, enfatizzare l'iterazione J2ME-PHP.

Quest'interazione ci permette di trasferire tutta la potenza di PHP su qualsiasi dispositivo che supporti J2ME. Nel nostro caso, infatti, se non avessimo utilizzato PHP lato server avremmo dovuto memorizzare i codici di quasi 8000 comuni sul cellulare, con tutte le inefficienze del caso.

Alessandro Lacava

DISCLAIMER

Attenzione!! Il CF generato corrisponde alle regole del D.M. del 12.3.1974. Non si può avere l'assoluta certezza dell'esattezza del codice perché, nel raro caso in cui il sistema genera due codici identici (caso di due persone con nome e cognome molto simili, nati lo stesso giorno, nello stesso comune), il Ministero delle Finanze provvede opportunamente alla sostituzione di uno di essi. Quindi ci sentiamo obbligati a sottolineare che il CF generato da questa applicazione può essere usato solo per verifica o per consentire un'attribuzione temporanea. L'autore dell'articolo e l'editore della testata declinano ogni responsabilità per eventuali danni di qualsivoglia natura subiti in seguito all'uso improprio di questa applicazione.

JAVA Controlla anche il tempo

Con Quartz della OpenSymphony progettiamo applicazioni in grado di eseguire operazioni ripetitive in momenti ben precisi e predeterminati dal programmatore

REQUISITI

Conoscenze richieste
J2SE

Software
J2SE SDK, Quartz 1.4.5

Impegno

Tempo di realizzazione

Avete mai pensato cosa rappresenta il sistema operativo al livello di astrazione più alto visto dall'utente?
Se pensate a Windows o a Linux in puri termini di applicazione, vi accorgerete che utilizzare un sistema operativo significa in una qualche misura fargli compiere una serie di operazioni ordinate secondo un qualche criterio logico.
Scendendo a basso livello, questa organizzazioni in "task" che devono essere eseguiti in maniera ordinata è persino più evidente.
Questo tipo di organizzazione in task schedulati per essere eseguiti ad intervalli di tempo precisi, oppure in relazione ad eventi specifici può essere riportata anche nelle nostre applicazioni Java.
Un esempio banale per quanto utile è quello di un'applicazione che ripetitivamente ad intervalli di tempo precisi controlli una casella di posta elettronica per vedere se c'è posta in arrivo. Gli esempi su applicazioni di scheduling si sprecano. Pensate a un task che il primo giorno di ogni mese sia programmato per eseguire ad una certa ora un backup di sistema, oppure un'applicazione che ogni settimana spedisca via email le statistiche di accesso ad un sito web. Insomma tutto ciò che coinvolge operazioni ripetitive nel tempo, può essere gestito tramite uno scheduler e tramite dei task. Lo scheduler si occuperà di avviare un task quando il suo timer coinciderà con la scadenza programmata dall'utente.

SCHEDULING STANDARD

Un primo livello di scheduling è quello supportato dal sistema operativo. Tipicamente ogni sistema operativo contiene un meccanismo che consente di attivare una qualche applicazione in un momento ben determinato ed indipendentemente dall'intervento dell'utente. In ambiente Linux esiste il demone: *crond*, che permette di schedulare le operazioni ad esempio. Cron controlla l'elenco dei task da eseguire e gli intervalli di tempo in cui eseguirli analizzando un file di testo contenuto tipicamente nella directory */etc/crontab*. Il file */etc /crontab* consiste di una serie di *"entry"* la cui sintassi è molto simile a quella che segue:

```
22 4 * * 0 root /root/task.sh
```

I primi due numeri riguardano il minuto in cui il task deve essere eseguito, poi l'ora, il giorno del mese, il mese dell'anno e il giorno della settimana. Poi viene indicato l'utente che ha i permessi per eseguire una data applicazione e infine il comando stesso. In questo modo sotto Linux possiamo eseguire determinati task in maniera precisa e schedulata. Sui sistemi Windows esiste un modo più "visuale" di inserire task da far eseguire al sistema. La cartella *"Operazioni pianificate"* consente di aggiungere un nuovo task, decidendo l'orario e direttamente il file da eseguire. Ogni sistema operativo ha dei suoi metodi particolari per schedulare dei task. A noi però interessa farlo programmaticamente con un nostro programma Java. In questo ambito esistono diverse librerie che possono venirci in aiuto.

SCHEDULING JAVA

Java supporta nativamente lo scheduling tramite le classi standard *Timer* e *TimerTask* che ci consentono di raggiungere lo scopo desiderato senza dover scomodare altre librerie esterne. La classe *TimerTask* conterrà il codice che vogliamo eseguire.
Dobbiamo prendere la nostra classe, estendere *TimerTask* e scrivere all'interno del metodo *run()* il codice che ci interessa.

```
import java.util.* ;
public class MioTask extends TimerTask  {
```

```
String stringa;
public MioTask(String stringa) {
   this.stringa=stringa;
}

public void run() {
   System.out.println(stringa);
}
}
```

Questa classe non è altro che un *Thread* (infatti *TimerTask* implementa l'interfaccia *Runnable*) che andremo a schedulare per essere eseguita dopo un certo tot di tempo.

```
Timer t=new Timer();
MioTask mt=new MioTask("Il tuo task che si attiva");
t.schedule(mt,5000);
```

In questo modo la classe *MioTask* andrà in esecuzione esattamente 5 secondi dopo la chiamata del metodo *schedule()*. Queste due classi nella maggior parte dei casi bastano alle nostre applicazioni. In altri casi invece abbiamo bisogno di un controllo migliore dei task schedulati e per questo dobbiamo rivolgerci ad altre librerie.

LA SOLUZIONE SEMPLICE

L'idea è utilizzare sistemi che da un lato possano essere eseguiti in modo standalone sul sistema, dall'altro esportino interfacce che ci consentano di programmare uno scheduler da un'applicazione Java. Una fra le librerie più interessanti è *JCron,* uno scheduler creato appositamente per Windows NT/2000 che lo usa come un vero e proprio servizio di sistema. Oltre a permettere di schedulare applicazioni Java questo scheduler, tramite un'interfaccia JNI, può gestire applicazioni native. Per il momento la versione 1.0 di JCron permette di schedulare applicazioni soltanto su sistemi Windows, il suo uso è quindi limitato ad un solo sistema operativo, anche se nella versione 2.0 è previsto un porting che lo renderà cross platform. Un altro scheduler degno di nota è *JCrontab.* Questo è uno scheduler leggermente più complesso e completo del precedente ed ha la caratteristica di essere scritto totalmente in Java, quindi portabile su ogni sistema operativo. JCrontab permette di eseguire Thread, classi, metodi, EJB, programmi nativi e quant'altro sia possibile. Tutto questo utilizzando un file di configurazione conforme al formato di crontab. Inoltre permette di salvare su ogni tipo di DataSource i job schedulati. Un ottimo scheduler, testato anche su Tomcat, Resin, Jetty e JBoss con risultati interesanti. Come ogni buon tool che si rispetti per usarlo nei nostri programmi servono due o tre righe di codice

```
// Istanziamo una classe di JCrontab utilizzando il
                                metodo statico getInstance()

Crontab crontab = Crontab.getInstance();
// Passiamo come argomento al metodo init il file di
     proprietà dove sono descritti tutti i job schedulati.

crontab.init("propertyFile", 60);
```

Dopo aver invocato il metodo *init()* JCrontab è avviato e, se non ci sono problemi nel file di configurazione, comincerà a schedulare i task inseriti per l'esecuzione. Per questo articolo abbiamo scelto però di utilizzare Quartz della OpenSimphony, che esporta alcune caratteristiche decisamente interessanti e che lo rendono uno dei migliori scheduler opensource scritti in Java.

QUARTZ

Quartz è uno scheduler scritto totalmente in Java, rilasciato con licenza opensource. Come le altre librerie è possibile integrarlo in un nostro progetto oppure farlo partire in modalità standalone. Una feature molto interessante di Quartz è la possibilità di salvare i task schedulati in maniera permanente. Tipicamente quando si utilizza la libreria standard di Java riguardante i Timer i task possono essere schedulati, ma quando il computer viene riavviato viene chiusa la JVM e le informazioni riguardanti i Timer instanziati vengono persi. Con Quartz è possibile, invece, salvare i vari task schedulati all'interno di veri e propri database e recuperarli al successivo riavvia della macchina

IL MIO PRIMO JOB

Lo schema di funzionamento per quanto riguarda lo scheduling è molto simile a quello utilizzato dalle API Java che abbiamo fin qui visto. Con Quartz infatti dobbiamo prima di tutto implementare un'interfaccia Job e definire il metodo execute() (esattamente allo stesso modo in cui con le classi *Timer* definivamo il metodo *run()* del *TimerTask*)

```
import org.quartz.*;
public class SimpleJob implements Job {
    public SimpleJob() {
    }

    public void execute(JobExecutionContext context)
      throws JobExecutionException
    {
```

```
        System.out.println("SimpleJob in esecuzione");
    }
}
```

Il passo successivo è instanziare il vero e proprio scheduler e inserirlo.

```
import org.quartz.impl.*;
import org.quartz.*;
import java.util.*;
public class PrimoEsempio {
  public static void main(String a[]) throws Exception {
    SchedulerFactory schedFact = new
                  org.quartz.impl.StdSchedulerFactory();
    Scheduler sched = schedFact.getScheduler();
    sched.start();
    JobDetail jobDetail = new JobDetail("primoJob",
               sched.DEFAULT_GROUP,SimpleJob.class);
    SimpleTrigger trigger = new SimpleTrigger(
                  "aTrigger",sched.DEFAULT_GROUP,new
                                  Date(),null,0,0L);
    sched.scheduleJob(jobDetail, trigger);
    Thread.currentThread().sleep(4000);
    sched.shutdown();
  }
}
```

Analizziamo ora insieme come abbiamo inserito questo primo Job. Prima di tutto otteniamo una *SchedulerFactory,* grazie alla quale poi otteniamo un'istanza dello scheduler. Una volta ottenuta questa istanza, con il metodo *start()* facciamo partire il vero e proprio scheduler. Quest'ultimo incomincia a controllare i task da eseguire. Creiamo un'istanza di *JobDetail*, una classe che serve per contenere diverse informazioni riguardanti il Job e la sua esecuzione. Infatti nel costruttore passiamo un nome che identifica il Job che vogliamo schedulare, il gruppo di scheduling e la vera e propria classe da eseguire. In seguito dobbiamo creare un trigger, una classe che serve a definire in quali situazioni deve partire il nostro Job. Per ora utilizziamo la classe *SimpleTrigger,* molto simile al Timer delle API Java standard, pur sapendo che esistono altri modi in Quartz per definire questo trigger.

Il *SimpleTrigger* che abbiamo instanziato prende come parametro un nome con il quale lo identifichiamo, un gruppo di scheduling, una data d'inizio, la data di fine esecuzione, quante volte deve essere ripetuto e l'intervallo di esecuzione. In questo caso abbiamo settato a *"null"* il tempo di fine esecuzione del Job e le volte che deve essere ripetuto, in modo tale che il nostro programma venga eseguito una sola volta. Passiamo quindi a schedulare il tutto con la chiamata *scheduleJob()* dello scheduler. Aspettiamo quindi 4 secondi, tempo durante il quale il nostro Job viene richiamato, viene eseguito il suo metodo *execute()* e quindi vediamo a schermo la scritta *"SimpleJob in esecuzione"*. Infine per non lasciare in esecuzione lo scheduler lo terminiamo con il metodo *shutdown()*. Abbiamo quindi visto come un nostro Job può essere richiamato dallo scheduler. Oltre ad essere eseguito il Job ha a disposizione delle ulteriori informazioni riguardanti la sua esecuzione, tutte memorizzate nell'oggetto *JobExecutionContext,* che viene passato come argomento al metodo execute che dobbiamo implementare. Questo oggetto contiene numerose informazioni che possiamo ottenere attraverso dei semplici metodi *get*.

```
public void execute(JobExecutionContext context)
  throws JobExecutionException
  {
       try {
         // Descrizione del trigger che ha avviato il Job
         System.out.println(context.getTrigger(
                                   ).getDescription());
         // Tempo di partenza del Job
         System.out.println(context.getFireTime());
         // Nome del JobDetail
         System.out.println(context.getJobDetail(
                                     ).getFullName());
        // Descrizione del JobDetail
        System.out.println(context.getJobDetail(
                                   ).getDescription());
        // Data della precedente partenza di questo Job
        System.out.println(
                        context.getPreviousFireTime());
        // Data della prossima partenza per questo Job
        System.out.println(context.getNextFireTime());
        System.out.println("SimpleJob in esecuzione.");
        // Settaggio del risultato ottenuto da questo Job
        context.setResult("Sono riuscito a scrivere");
  }
  catch(Exception e) {
      context.setResult("Errore nell'esecuzione
                                 :"+e.toString());
  }
  // Stampa del risultato a schermo
  System.out.println(context.getResult());
}
```

Analizzando i dati che contiene l'oggetto *JobExecutionContext* possiamo quindi scrivere una diversa implementazione per il nostro Job. Ad esempio pensiamo ad un programma che deve fornire un risultato in base anche a quando è stato lanciato l'ultima volta, o in base a determinate informazioni che sono memorizzate nella descrizione del *JobDetail* o del *Trigger.* Per comunicare altre importanti informazioni riguardanti l'esecuzione al nostro Job possiamo utilizzare una sorta di Map che troviamo all'interno della semplice istanza di JobDetail. In questo modo possiamo comunicare parametri e altre informazioni che possono modi-

SUL WEB

Ecco le diverse implementazioni di scheduler in Java e i link alle homepage.

Quartz
http://www.opensymphony.com/quartz

JCron
http://p.clark.home.mindspring.com/jcron

JCronTab
http://jcrontab.sourceforge.net

Timer API Java
http://java.sun.com/developer/TechTips/2000/tt0530.html#tip2

ficare l'esecuzione del nostro Job.

```
JobDetail jobDetail = new JobDetail(
  "primoJob",sched.DEFAULT_GROUP,SimpleJob.class);
jobDetail.getJobDataMap().put("num",numcell);
jobDetail.getJobDataMap().put("testo",testoreminder);
```

Ora abbiamo inserito nel *JobDataMap* due diverse coppie chiave/valore, da usare ad esempio per schedulare testo da inviare a differenti utenti attraverso un sms.

```
JobDataMap map = context.getJobDetail(
                                   ).getJobDataMap();
String num=map.getString("num");
String testo=map.getString("testo");
inviaSms(num,testo);
```

CRON TRIGGER

Vediamo ora come sia possibile definire dei trigger in Quartz, utilizzando la stessa sintassi che viene utilizzata per definirli all'interno di cron. Definiamo due semplici trigger che attiveranno un programma in diversi momenti della giornata periodicamente.

```
String espressioneStartup="0 0 9 ? * MON-FRI";
String espressioneShutdown="0 0 18 ? * MON-FRI";

CronTrigger ct1=new CronTrigger(
    "ct1",,sched.DEFAULT_GROUP,espressioneStartup);

CronTrigger ct2=new CronTrigger(
  "ct2",,sched.DEFAULT_GROUP,espressioneShutdown);
```

I due trigger creati riguardano l'esecuzione di un nostro programma ogni mattina alle 9 dal lunedì al venerdì (*espressioneStartup*) e ogni pomeriggio alle 18 dal lunedì al venerdì *(espressioneShutdown)*.
Potremmo utilizzare una cosa del genere per automatizzare magari qualcosa che facciamo ogni volta che entriamo e usciamo dal lavoro.

FEATURES AVANZATE

Un aspetto molto interessante di questa libreria opensource è il fatto di poter utilizzare dei listener che ci avvertono quando eventi come Trigger e Job vengono scatenati. Se vogliamo avere informazioni riguardanti un JobDetail eseguito dobbiamo implementare l'interfaccia JobListener. In questa ci sono metodi che vengono richiamati quando un JobDetail sta per essere eseguito oppure quando è già stato eseguito. Allo stesso modo esiste *TriggerListener* che è l'interfaccia analoga riguardante gli eventi di un Trigger. Grazie a questo sistema di notifica oltre a schedulare task e applicazioni possiamo anche definire dei listener globali nelle nostre applicazioni per motivi logici o per motivi riguardanti la particolare implementazione di un metodo di business. Un'altra feature molto importante che riguarda Quartz è il fatto di poter scegliere come salvare le informazioni riguardanti Job, Trigger e quant'altro. Queste scelte però non devono essere fatte direttamente da codice, ma mediante il file di configurazione di Quartz, dove possiamo indicare diverse caratteristiche che il nostro scheduler dovrà rispettare. Ad esempio per utilizzare il sistema di memorizzazione volatile su RAM dobbiamo scrivere nel file di configurazione la seguente riga

```
org.quartz.jobStore.class =
                          org.quartz.simpl.RAMJobStore
```

Questo metodo di memorizzazione è quello più veloce e performante, ma chiaramente quando il sistema viene riavviato vengono perse tutte le informazioni relative ai nostri task. Un altro metodo di memorizzazione è quello riguardante il database, ovvero tutte le strutture dati che vengono create dal nostro scheduler vengono salvate su database e quindi un riavvio del sistema non compromette i dati salvati. Per utilizzare il database dobbiamo chiaramente creare una tabella indicando nel file di configurazione anche il prefisso standard di tutte le tabelle (ad esempio *QZ_nome*, *QZ_numcell*). Poi possiamo decidere due diverse tipologie di connessione al database, TX o CMT. TX viene usato nel caso in cui le connessioni al database devono supportare le transazioni, mentre con una connessione CMT lasciamo la gestione delle transazioni ad altre entità (ad esempio all'application server che stiamo utilizzando con Quartz).

CONCLUSIONI

Come abbiamo potuto vedere, Quartz è uno strumento abbastanza potente da includere nelle nostre applicazioni. Inoltre il fatto di poter schedulare dei task e salvarli è di notevole importanza per molti nostri programmi. In questo modo infatti non dobbiamo spaventarci se un server che abbiamo viene riavviato perché comunque tutte le informazioni schedulate sono presenti nel database. Inoltre come potete vedere dal sito ufficiale (*http://www.opensymphony.com/quartz/*), Quartz viene utilizzato da molte compagnie famose in diversi sistemi informatici.
Insomma un ottimo tool da conoscere e da utilizzare.

Federico Paparoni

L'AUTORE

L'autore, Federico Paparoni, può essere contattato per suggerimenti o delucidazioni all'indirizzo email federico.paparoni@javastaff.com

eXtreme Programming

Gestire il ciclo di sviluppo del software non è un'operazione banale. Spesso i costi dovuti a mantenimento e modifiche superano quelli della realizzazione. Ecco un metodo per evitare che questo avvenga

In letteratura esistono molti modelli che descrivono il ciclo di vita del software. Considerando ad esempio il classico (e ormai superato) modello a cascata di Royce possiamo individuare in sequenza le seguenti fasi: *Analisi, Progettazione, Realizzazione, Manutenzione.* Importando in una linea del tempo i costi, in termini di ore impiegate, di ogni fase, ci renderemmo conto che la manutenzione del software (intesa come correzione bug, adattamento e miglioramento) pesa in maniera sproporzionata su tutto il progetto. Considerando altri ambiti progettuali cui possa essere applicata la stessa suddivisione dei task individuata per il software (pensiamo a progetti edilizi o meccanici) è naturale chiedersi come mai la manutenzione non arrivi mai ad essere così incidente sui tempi globali. Perché la progettazione in ambiti diversi dal software può essere precisa e sicura mentre nel nostro ambito l'insorgenza dei bug, le modifiche in corsa, l'imprecisione del disegno iniziale sono considerati aspetti normali? La risposta più immediata che ci si può dare è che si tratti di un difetto congenito del software che nasce dal suo essere un prodotto complesso, astratto e poco rappresentabile e che difficilmente può essere tenuto sotto controllo nei minimi dettagli.

REQUISITI

Conoscenze richieste
Basi di .NET

Software
Visual Studio.NET

Impegno

Tempo di realizzazione

SOLUZIONI AL PROBLEMA

Prendiamo atto di quella che sembra essere una stortura peculiare del software e con questa chiave di lettura andiamo ad interpretare come nel tempo varie soluzioni siano state avanzate per aggirare i succitati problemi. Pensiamo ad esempio all'introduzione della OOP, alla creazione di controlli sofisticati nei compilatori e negli ambienti di run-time: la tipizzazione delle variabili, la memoria gestita, la gestione strutturata delle eccezioni fino ad arrivare ai puntatori a funzione tipizzati di .NET (*delegate*). L'*eXtreme Programming* può essere visto sotto questo aspetto come l'ennesimo passo verso la creazione di codice sicuro e controllato anche se i dettami originali dell'XP vanno molto oltre questa sia pur importante caratteristica. Kent Beck il fondatore dell'*XP movement* dice che XP è un insieme di pratiche per lo sviluppo del software basate sui valori della semplicità, della comunicazione con il committente e tra i membri del team di sviluppo, del feedback e del coraggio! Come queste asserzioni non rimangano semplici slogan programmatici sarà chiaro scorrendo il resto dell'articolo.

IL CLASSICO PATTERN DI SVILUPPO DELL'XP

Vediamo sinteticamente quali sono le fasi principali che lo sviluppo software aderente ai dettami dell'XP, prevede:

- individuazione del problema;
- scomposizione del problema nelle sue unità fondamentali;
- creazione di set di test per ogni singola unità di codice;
- creazione vera e propria dell'unità di codice;
- testing attraverso i set di test;
- refactoring del codice.

Il processo di sviluppo del software, arrivato a questo punto, ben lungi dall'essere terminato riprende e si ripete più volte cercando di arrivare all'obiettivo per approssimazioni progressive.

I TEST

La più grande novità introdotta dall'XP è quindi l'introduzione dei test che le singole unità devono poter superare (per unità possiamo intendere la parte più elementare in cui può essere suddivisa un'elaborazione complessa). Una unità risponderà alle esigenze dell'applicazione non quando compirà le funzioni necessarie all'applicazione ma quando supererà tutti i test a cui verrà sottoposta. Di conseguenza, il soddisfacimento dei requisiti di funzionamento richiesti dall'applicazione diventa incidentale, i requisiti stessi sono incarnati nei test. Vediamo, con un esempio, cosa significha a livello pratico un approccio di questo tipo. Immaginiamo di essere ad una riunione con il nostro committente che ci sta spiegando quali dovranno essere le funzionalità dell'applicazione. Tra le tante essa dovrà poter restituire, dato il CAP, il nome della città corrispondente completo di provincia di appartenenza. Una richiesta di questo tipo è già di per sé un test. Il committente infatti non ci sta chiedendo di far funzionare il nostro codice in un certo modo ma che *"a fronte di certi dati inseriti dovrà risultare un certo risultato"*. Quindi formalizzando il test in questo modo: *"Inserito il cap 35100 dovranno risultare la città di Padova e la provincia PD"*. Il ruolo del committente nella creazione dei test non è assolutamente da sottovalutare: è il committente l'unico che sa che cosa l'applicazione debba fare, quali siano i casi particolari che l'applicazione dovrà gestire. Grazie alla stesura, assistita dallo sviluppatore, dei test, il committente ha uno strumento per "raccontare" la sua idea di applicazione e, successivamente, per verificare se l'applicazione risponda effettivamente alle sue richieste. Questo tipo di approccio è comunemente detto test driven cioè guidato dai test.

IL REFACTORING

Un altro concetto inusuale nello sviluppo software approcciato in modo tradizionale è senz'altro quello del refactoring. Di solito nella previsione dei tempi di sviluppo non viene dato spazio a questa pratica mentre l'XP ne fa una delle sue fasi di punta. Il *refactoring* è l'operazione attraverso cui viene migliorato il design dell'applicazione. Non un semplice make-up ma un cambiamento strutturale. Chi non ha molta esperienza nella programmazione si potrebbe chiedere che senso abbia prevedere già una fase dove vada modificata la struttura dell'applicazione quando molto più sensatamente si potrebbe disegnarla in modo corretto già all'inizio. Purtroppo all'inizio del progetto la conoscenza del problema è spesso approssimativa sia da parte dell'analista sia del committente stesso (quanti processi aziendali vengono formalizzati proprio in occasione della creazione dell'applicazione che deve supportarli!!). Come se ciò non bastasse il committente stesso è incline a richiedere frequenti cambiamenti in corso d'opera. Alla luce di questi fattori è più facile capire come il disegno iniziale, curato, ordinato e pulito, venga pian piano alterato fino a perdere chiarezza e manutenibilità. È qui che subentra il *refactoring* a ridare coerenza ad un'architettura ormai spenta. Per un'approfondimento sulle pratiche di *refactoring* vi invito a consultare la bibliografia. Benché le pratiche di *refactoring* siano molto ben codificate (in teoria seguendole pedissequamente non dovrebbe mai succedere che l'applicazione finisca per non funzionare) mettere mano all'architettura di un'applicazione è sempre un po' rischioso. Le unit test permettono un certo grado di sicurezza indicando immediatamente quando una routine sottoposta a *refactoring* non si sta più comportando come previsto. Ecco perché Kent Beck dice che l'XP ha tra i suoi valori fondanti il coraggio: bisogna sempre essere pronti a cambiare il codice attraverso il *refactoring* perché comunque attraverso i test sapremo se le modifiche introducono bug.

EMERGONO I PRIMI BUG

Simuliamo ancora un caso reale. La nostra applicazione comincia a crescere e com'è naturale iniziano ad essere riscontrati i primi bug. Non ha importanza in ambito XP se i bug nascono a causa del refactoring, a causa di modifiche chieste dal committente, a causa di codifica errata o a causa di codifica concorrente da parte di più sviluppatori sullo stesso software. L'XP ci chiede di fare un'unica cosa: se già non è stato fatto, va scritto un test che verifichi il bug cioè un test che fallisca a causa del bug. In seguito va corretto il codice e a questo punto il test deve riuscire.

RIUSCITA E FALLIMENTO DEI TEST: COROLLARIO ALLA REGOLA GENERALE

A parità di codice un test che fallisce alla prima esecuzione deve fallire sempre mentre un test che riesce alla prima esecuzione deve riuscire sempre. Il presupposto che sottostà a questa regola è che l'oggetto a cui vengono sottoposti i test deve avere uno stato consi-

I TUOI APPUNTI

stente e certo. Chiaramente se il nostro test va a verificare l'algoritmo che fa la somma di due numeri, lo stato consistente sarà sempre verificato a meno di variazioni nelle regole basilari della matematica. Ma pensiamo ad esempio ad un test che verifichi il numero di file in una cartella del file system; il suo risultato dipenderà da quanti file sono presenti e quindi il suo stato iniziale non è certo ma dipende dal contesto. Si potrebbe obiettare che l'esempio è poco significativo e facilmente risolvibile ma pensate a quali implicazioni potrebbe produrre un test su un'istanza di una classe complessa (es. la classe cliente che deve esporre una serie di proprietà già valorizzate in un certo modo) o su un DAL *(Data Access Layer)* che presuppone che il contenuto delle tabelle del database sia di un certo tipo. (La trattazione di questi argomenti va oltre gli scopi dell'articolo perciò per maggiori informazioni vi invito a vedere in bibliografia i riferimenti ai due articoli di MSDN Magazine sui *Mock Object* e sul testing dei *Data access Layer*).

Il framework NUnit è liberamente scaricabile all'indirizzo www.nunit.org dove si possono trovare varie risorse e documentazione.

NUNIT

Veniamo finalmente alla pratica. *NUnit* è un framework free per l'implementazione dei test in .NET ed è l'ultimo nato di una lunga serie di *framework Unit* (il più famoso è senz'altro *JUnit* per Java). I test possono essere scritti nel linguaggio .NET preferito in quanto il cuore del framework risiede in un'unica istruzione, disponibile in ogni linguaggio, che è il costrutto *Assert* (asserzione, affermazione) e in una serie di attributi. *NUnit* viene distribuito con degli esempi e con una completa serie di test cui può sottoporre se stesso (è infatti distribuito con codice sorgente in C#). L'installazione è semplice e guidata attraverso un programma di setup. Una volta installato il framework troveremo nel menu programmi un eseguibile *(NUnit-GUI)* e una serie di esempi che ci introdurranno velocemente all'utilizzo degli *unit test.*

- **EXTREME PROGRAMMING ADVENTURE IN C#** ***Ron Jeffries*** **(Microsoft Professionale)**

IMPLEMENTAZIONE DEI PRIMI TEST

Creiamo una libreria di classe che chiameremo *OperazioniMatematiche*. La libreria avrà solamente quattro metodi che chiameremo naturalmente *Addizione*, *Sottrazione*, *Moltiplicazione* e *Divisione*. Vediamo il primo codice C# per il metodo *Divisione.*

```
public decimal  Divisione(decimal Dividendo,decimal
                                          Divisore)
{  return Dividendo/Divisore;
}
```

Ora passiamo alla creazione dei test. I test possono risiedere esternamente all'assembly in modo da non inserire codice funzionalmente inutile all'interno del nostro codice. Quindi creiamo in un progetto separato, un'altra libreria di classe, in cui inseriremo due riferimenti:

- il riferimento al nostro assembly *OperazioniMatematiche;*
- il riferimento al framework *NUnit* che sarà disponibile tra i componenti .NET.

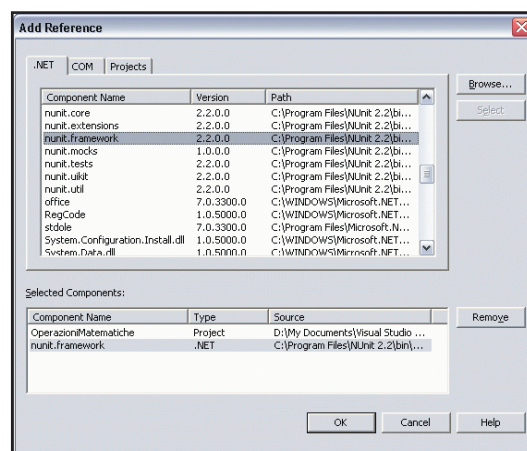



Fig. 1: ***Il primo passo è aggiungere i riferimenti all'assembly e al framework NUnit***

La classe di test avrà a questo punto tutti i riferimenti necessari per cominciare il suo lavoro. Iniziamo ad indicare a *NUnit* che la nostra classe *UTSuOperazioniMatematiche* ospita dei test da sottoporre all'assembly. Per far ciò è sufficiente inserire l'attributo *TextFixture* (negli esempi userò sempre la notazione estesa) in testa alla classe.

```
[NUnit.Framework.TestFixture]
public class UTOperazioniMatematiche
```

Ora dobbiamo semplicemente scrivere un test. Anteponiamo al codice del test l'attributo corretto *(Test)* e poi instanziamo il nostro assembly *OperazioniMatematiche*. L'ultima riga imposta l'asserzione che, in questo caso, deve essere vera *(IsTrue)* in quanto 12/3 deve dare 4.

```
[NUnit.Framework.Test]
public void TestaDivisione()
{ OperazioniMatematiche testaDivisione=new
                           OperazioniMatematiche();
   NUnit.Framework.Assert.IsTrue(
```

```
        testaDivisione.Divisione(12,3)==4);
}
```

ESECUZIONE DEI TEST

A questo punto, preparata l'infrastruttura, possiamo procedere all'esecuzione dei test.
Per la verifica dei test *NUnit* offre una modalità console e una modalità *GUI* disponibile nel menu *Programmi*. Aperto il programma che offre la modalità *GUI* dovremo indicare ad *NUnit* che vogliamo creare un nuovo progetto di testing (menu file – *New project*). Consiglio di tenere uniti i file di progetto di *NUnit (*.nunit)* e i file di progetto delle classi di test per una questione di praticità d'uso. Per maggiore semplicità è anche possibile indicare direttamente il file di progetto .NET senza creare il file di progetto *NUnit*. Una volta creato il progetto di testing va caricato l'assembly con i test veri e propri (nel nostro caso *UTOperazioniMatematiche*) attraverso il menu *Project – Add assembly*. *NUnit-GUI* è pronto per effettuare I test in modo automatico. Dovreste infatti visualizzare sotto forma di pallini grigi i vari elementi dell'assembly identificati dai vari attributi di *NUnit*. Premendo il tasto *Run* i test vengono eseguiti e compare un pallino verde in luogo di un test riuscito, e un pallino rosso quando un test fallisce. In caso di fallimento di un test verrà prodotta anche la descrizione dell'errore di *NUnit*.

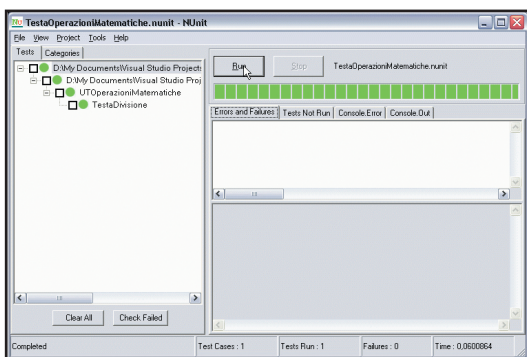



Fig. 2: *NUnit restituisce graficamente l'esito di un test*

ATTRIBUTI E ASSERZIONI

NUnit mette a dispozione molti vari attributi che possono essere utilizzati nelle classi di test. Abbiamo già visto gli attributi principali *TestFixture* e *Test* a cui dobbiamo aggiungere:

- **TestFixtureSetUp** e **TestFixtureTearDown** – indicano che il codice va eseguito rispettivamente prima e dopo qualunque altro test contenuto all'interno di una classe TestFixture. L'utilità di questi due attributi è intuitivo: servono a creare l'ambiente consistente di cui necessitano i test per funzionare (ad es. *TestFixtureSetUp* potrebbe creare un file con un certo contenuto su cui poi i vari test opereranno mentre *TestFixtureTearDown* si occuperà di cancellare il file).
- **ExpectedException** – indica che il test in oggetto deve sollevare un'eccezione: ad es. un test sulla divisione che attribuisca al divisore il valore 0 dovrà attendersi un errore del tipo *System.DivideByZeroException*

```
[NUnit.Framework.Test]
[NUnit.Framework.ExpectedException(typeof(
                    System.DivideByZeroException))]
public void TestaDivisionePerZero()
{  testaDivisione.Divisione(12,0);
}
```

Oltre a questi esistono anche altri attributi meno minori. Le asserzioni, di cui abbiamo visto un solo esempio, possono essere del tipo di condizione:

- *IsTrue*
- *IsFalse*
- *IsNull*
- *IsNotNull*

Del tipo di confronto:

- **AreEqual** (per confronti tra tipi valore)
- **AreSame** (per confronti tra tipi riferimento)
- Esiste inoltre un metodo **Assert.Fail** che permette di far fallire il test senza bisogno di verificare nessuna asserzione.

CONCLUSIONI

L'idea di mettere sotto controllo il codice attraverso altro codice è già di per sé avvincente ma il fatto che questa tecnica permetta veramente di avere un controllo così stretto sul codice dovrebbe convincere chiunque della bontà dell'approccio XP. Oltre alla consultazione della bibliografia (purtroppo quasi esclusivamente in inglese) vi invito caldamente a rifarvi al filone dei programmi open source in C# (o VB.NET) che molto spesso sono distribuiti con unità di test già implementate. Un caso reale è dato dall'ottimo feed aggregator RSSBandit che potete scaricare da Source Forge in codice sorgente con un completo set di test per NUnit.

Gianluca Negrelli

SUL WEB

http://msdn.microsoft.com/msdnmag/issues/04/10/NMock

http://msdn.microsoft.com/msdnmag/issues/04/04/ExtremeProgramming

http://www.ugidotnet.org/articles/articles_read.aspx?ID=85

http://msdn.microsoft.com/msdnmag/issues/05/06/UnitTesting/default.aspx

http://msdn.microsoft.com/msdnmag/issues/05/02/ExcelUnitTests/default.aspx

L'AUTORE

Si occupa di analisi architetturale e programmazione di applicazioni enterprise winform e web con framework .NET sia in C# che in VB. Si occupa di sistemi di accesso e gestione dati di database in ambiente Windows.
Per contattarlo:
gianluca.negrelli@gmail.com

AspectJ aiuta Java a migliorare

Iniziamo a lavorare con AOP e Java, facendo la conoscenza di un nuovo paradigma di programmazione che offre la possibilità di risolvere in maniera concisa alcuni problemi tipici di molti software

Spesso, in nello sviluppo di un sistema informatico, diventano preponderanti aspetti non strettamente legati agli obiettivi principali per cui il software viene sviluppato. Il codice business è "annegato" in altro codice per la gestione dei file di log, delle eccezioni, dei controlli sugli accessi, ecc... Il codice per la gestione di questi aspetti tipicamente non è presente in una singola classe ma è distribuito, e questo spesso rende complesso il riuso e la manutenzione. L'*Aspect Oriented Programming* si pone come scopo quello di offrire un nuovo modello di programmazione che permetta di fattorizzare questi aspetti che non sono responsabilità di alcuna classe ma la cui implementazione incide su un gran numero di esse. AOP si integra con paradigmi consolidati come quello della programmazione ad oggetti per lo sviluppo dei requisiti più puramente business. AOP si basa su concetti diffusi e riconosciuti e numerose sono le "estensioni" AOP per Java, tra le quali *AspectJ* argomento dell'articolo. Con *AspectJ* è sostanzialmente possibile lanciare l'esecuzione di codice Java nel momento in cui il flusso di esecuzione raggiunge alcuni punti definiti, quali l'invocazione di un metodo o di un costruttore, l'assegnazione di un valore ad un attributo, ecc... Una sorta di *"interrupt"* legato al flusso di esecuzione del programma. In coincidenza di una certa istruzione si scatena un evento che lancia altro codice. I punti del sorgente a cui si desidera agganciare l'esecuzione di altro codice, sono chiamati *"joinpoint"*. Si definiscono poi come *"pointcut"*, insiemi di *joinpoint*, combinati tra loro anche da operatori logici ed infine si definiscono gli *"advice"* che specificano il codice da eseguire al verificarsi di un particolare *pointcut*. L'insieme di *pointcut*, *joinpoint* e *advice* prende il nome di *"aspetto"*.

UNA CLASSE A CUI APPLICARE AOP

Create un nuovo progetto con la voce di menu *File > New > Project...* Nella finestra che

COME INIZIARE

Gli i esempi che accompagnano l'articolo saranno sviluppati con Eclipse, IDE open source liberamente disponibile, e AJDT *(AspectJ Development Tool)* un relativo plug in che offre la possibilità di gestire programmi AOP con *AspectJ.* La seguente procedura illustra come recuperare gli strumenti per realizzare gli esempi che accompagnano l'articolo.

1. Recuperare l'ultima versione di Eclipse all'indirizzo http://www.eclipse.org/downloads/index.php. Scaricare l'ultima versione e installarla.

2. Lanciare Eclipse e configurarlo in modo che scarichi automaticamente da Internet il plug in AJDT. Scegliere la voce di menu *Help > Software Updated > Find and Install...* Nella finestra che appare selezionare "Search for new features to install" e "Next".

3. Nella finestra di dialogo premere *"New Remote Site..."*. Nella finestra che appare digitare nel campo *"name"* "*AJDT Update Site*" e come URL "http://download.eclipse.org/technology/ajdt/31/dev/update"

4. Aprire il nodo *"AJDT Update Site"* che appare, e selezionare "*AspectJ*". Premere "*Next*"

5. Viene mostrata una finestra che elenca gli elementi installabili. Selezionare "*Eclipse AspectJ Development Tools*" e premere "*Next*".

6. Nella pagina successiva accettare la licenza.

7. Al termine premere "*Finish*" e attendere che l'installazione venga completata.

8. Riavviate Eclipse.

si apre scegliete *AspectJ > AspectJ Project*. Nella finestra immettete il nome di progetto *"aspectj"*. Nella finestra successiva accettate il layout delle cartelle proposto. Premere quindi *"finish"*. Anche nella finestra seguente lasciate le impostazioni di default. Cliccate la voce di menu *File > New > Class*. Come nome della classe immettete *"Hello"* e come package *"ioprogrammo"*. Il programma da digitare è una variante del classico *"Hello World"*.

```
package ioprogrammo;
public class Hello
   public void sayHello()
   {
      System.out.println("Hello");
   }
   public static void main(String[] args)
   {
        Hello h = new Hello();
        h.sayHello();
   }
}
```

Selezionate ora il menu *Run > Run...* e dalla finestra che si apre il tasto *"Search"* nella casella *"Main class"*. Selezionate la classe appena creata e premete *run*. Nella console, a dimostrazione dell'avvenuta esecuzione apparirà:

Hello

IL PRIMO ASPETTO

Scegliete ora *File > New > Other... > AspectJ > Aspect* per creare un aspetto. Nella finestra che si apre immettete come nome *"HelloAspect"* e come package *"ioprogrammo"*. Digitate il seguente codice:

```
package ioprogrammo;
public aspect HelloAspect
{
```

Così si definisce il package dell'aspetto come avviene per le classi, ed i relativi nome e modificatore di accesso.

```
pointcut pc1(): call(public void Hello.sayHello());
```

La precedente riga definisce un *pointcut*, (un insieme di *joinpoints*), cioè punti del programma, raggiunti i quali dal flusso di esecuzione, verrà eseguito il codice dell'aspetto. In particolare è stato definito un pointcut di nome *"pc1"* costituito dalla chiamata al metodo *public void sayHello()* della classe *Hello*.

Definiamo ora due advice, porzioni di codice da eseguire al raggiungimento del *pointcut*, che scatteranno prima e dopo il raggiungimento del *pointcut*.

```
   before():pc1()
```

HELLO WORLD IN TRE PASSI

Create una classe HelloWorld che esponga il metodo SayHello che non fa altro che stampare una stringa a schermo. Implementeremo un aspetto che intercetta le chiamate al metodo e stampa qualcosa prima e dopo di esso.

> UN NUOVO PROGETTO

1 Create un nuovo progetto con la voce di menu File > New > Project... Nella finestra che si apre scegliete AspectJ > AspectJ Project. Nella finestra immettete il nome di progetto "aspectj". Siamo pronti per creare il nostro primo aspetto.

> IL LAYOUT

2 Accettate il layout delle cartelle proposto. Premere quindi "finish". Nella finestra seguente lasciate le impostazioni di default. Cliccate la voce di menu File > New > Class. Come nome della classe usate "HelloAspect" e come package "ioprogrammo

> IL CODICE

```
package ioprogrammo;
public aspect HelloAspect {
   public void sayHello(){
      pointcut pc1():
      call(public void Hello.sayHello());
      before():pc1(){
         System.out.println("prima"); }
      after():pc1(){
         System.out.println("dopo");}
      }
}
}
```

3 Selezionate ora il menù Run > Run... e dalla finestra che si apre il tasto "Search" nella casella "Main class". Selezionate la classe appena creata e premete run. Se tutto è andato a buon fine nella console apparirà "prima" - "hello world" - "dopo"

```
    {
      System.out.println("prima");
    }
    after():pc1()
    {
      System.out.println("dopo");
    }
}
```

Ripetete la procedura di lancio descritta precedentemente. Nella console apparirà il seguente testo che dimostra come il codice dei due advice sia stato invocato prima e dopo il *joinpoint*.

prima
Hello
dopo

UN ESEMPIO PIÙ COMPLESSO

L'applicazione di esempio è un sistema per la prenotazione di biglietti per spettacoli cinematografici via internet. Le classi principali sono le seguenti.

- **Customer:** ha le responsabilità di scegliere lo spettacolo da acquistare e pagare i biglietti. Un cliente può scegliere di pagare con fidelity card solo se è stato abilitato dal cinema, magari perché ha visto un certo numero di spettacoli o perché ha aderito ad una campagna promozionale.
- **Movie:** è lo spettacolo stesso, con orario di inizio, titolo del film e disponibilità dei posti per lo spettacolo.
- **Transaction:** è la transazione economica. Memorizza l'intestatario della stessa, lo spettacolo ed il numero di posti acquistati e gli altri dati relativi alla transazione economica.

Customer richiede tramite un metodo statico di *Movie* l'elenco dei film tra cui scegliere quello per cui acquistare i biglietti. Identificato il film *Customer* richiama il metodo *openTransaction* di *Movie* che crea una nuova *Transaction* relativa all'acquisto di un certo numero di posti con un certo tipo di pagamento.
Customer paga tramite il metodo *pay()* di *Transaction* e al termine dell'operazione di convalida ed autorizzazione della transazione i biglietti sono infine acquistati.

IL SISTEMA DI ESEMPIO

Di seguito l'estratto delle classi che compongono questa applicazione.

```
package ioprogrammo;
public class Customer
{
  //Se questo cliente è un superCustomer.
  //ha il diritto di pagare con fidelity card
  private boolean isSuperCustomer;
  public Customer(boolean isSuperCustomer) {[...]}
    // azioni compiute dall'acquirente
    // per l'acquisto del biglietto.
    public void run()
    {
      //esempio acquisto con
      // carta di credito
      Transaction transaction;
      Movie[] shows =
      Movie.findShows();
      transaction = shows[2].openTransaction(
      this,Transaction.CREDIT_CARD_PAYEMENT, 2);
      transaction.pay();
      //acquisto con fidelity card
      transaction = shows[1].openTransaction(
      this,Transaction.FIDELITY_CARD_PAYEMENT, 2);
      transaction.pay();
    }
    public boolean isSuperCustomer() {[...]}
}
```

Il metodo principale della classe *Movie* è *openTransaction* che crea una nuova transazione relativa al film stesso.

```
package ioprogrammo;
import [...];
public class Movie
{
  [...]
public Movie(String title, int month, int date, int year,
                                    int hour, int minute) {//...}
  public String toString() {//..}
  //Restituisce un array di Movie
  // tra i quali Customer
  //può scegliere quello da acquistare.
  public static Movie[] findShows()
  {
     [...]
  }
  public Transaction openTransaction(Customer
            customer, int creditCardPayement, int seats)
  {
    //la transazione può essere aperta solo se
    //c'è almeno la disponibilità di posti richiesta.
    if (availability>=seats)
    {
      Transaction transaction = new Transaction(
```

```
            this, customer, seats, creditCardPayement);
        availability = availability - seats;
        return transaction;
      }
      throw new IllegalStateException("The requested
                  number of seats is not available.");
   }
   public void addSeats(int seats) {[...]}
   public String getTitle() {return title;}
}
```

La classe *Transaction* definisce lo stato della transazione che può essere *"non pagata"*, *"confermata"* se il sistema di pagamento ha autorizzato la transazione e *"abortita"* in caso contrario.

```
package ioprogrammo;
public class Transaction
{
   //tipi di pagamenti
   public static int CREDIT_CARD_PAYEMENT = 0;
   public static int FIDELITY_CARD_PAYEMENT = 1;
   //possibili stati della transazione
   public static int TRANSACTION_NOT_PAYED = 0;
   public static int TRANSACTION_CONFIRMED = 1;
   public static int TRANSACTION_ABORTED = 2;
   [...]
   public Transaction(Movie movie, Customer
      customer, int numberOfSeats, int payementType)
   {
      [...]
   }
   public Customer getCustomer() {[...]}
   //in base al tipo di transazione
   // delega la verifica del pagamento
   //ai metodi creditCardPayement()
   // e fidelityCardPayement()
   public void pay()
   {
     [...]
   }
   public int creditCardPayement()
   {
     //collegamento al sistema di
     //pagamento con carta
     //di credito
   }
   public int fidelityCardPayement()
   {
      //collegamento al back office
      // del cinema
      //per autorizzazione
      //pagamento con carta fedeltà.
   }
   public String toString()
   {
      [...]
```

```
   }
}
```

UN ASPETTO PER LA MISURAZIONE DEI TEMPI DI ESECUZIONE

Il primo problema dell'applicazione potrebbe emergere a seguito di segnalazioni di clienti che lamentano tempi di risposta per l'esito della transazione troppo elevati. Un primo passo potrebbe essere quello di aggiungere un aspetto che si occupi del monitoraggio dei tempi di esecuzione dei vari metodi.

```
package ioprogrammo;
import java.util.*;
public aspect LoggingAspect
{
   private Map methodToExecutionTime = new
                                    HashMap();
   pointcut allPublicMethods(): execution(public *
                         ioprogrammo..*(..));
   before():allPublicMethods()
   {
       String signature =
                thisJoinPoint.getSignature().toString
      methodToExecutionTime.put(signature, new
                Long(System.currentTimeMillis()));
   }
   after():allPublicMethods()
   {
      String signature =
               thisJoinPoint.getSignature().toString();
      Long startTime = (Long)
               methodToExecutionTime.get(signature);
      System.out.println(">>" + signature + " elapsed:
               " + (System.currentTimeMillis() -
         startTime.longValue()) + " milliseconds.");
    }
}
```

Questo aspetto definisce il pointcut *"allPublicMethods"* che incorpora, grazie all'uso delle wildcard, tutti le chiamate a metodi public di qualsiasi classe del package *ioprogrammo*. Prima dell'invocazione di qualsiasi metodo, l'advice estrae l'orario di sistema in millisecondi e lo pone in una *Map*, associandolo alla signature del metodo.
Dopo l'invocazione del metodo il secondo *advice* estrae nuovamente la data di sistema confrontandola con quella registrata dal precedente advice e stampando quindi il tempo trascorso. Questo aspetto mostra come sia possibile avere una misura dei tempi di esecuzione senza toccare minimamente il codice

presenti nelle classi. Lanciando l'applicazione ecco cosa appare nella console.

```
>>starting:void
        ioprogrammo.Customer.main(String[])
>>starting:void ioprogrammo.Customer.run()
>>starting:Movie[] ioprogrammo.Movie.findShows()
>>ending:Movie[] ioprogrammo.Movie.findShows()
        elapsed:80 milliseconds.
[...]
>>starting:int
        ioprogrammo.Transaction.creditCardPayement()
>>ending:int
        ioprogrammo.Transaction.creditCardPayement()
        elapsed:360 milliseconds.
>>ending:void ioprogrammo.Transaction.pay()
        elapsed:360 milliseconds.
[...]
>>starting:void ioprogrammo.Transaction.pay()
[...]
>>ending:void ioprogrammo.Transaction.pay()
        elapsed:10 milliseconds.
[...]
```

Da cui si evince che i metodi meno performanti siano proprio i due metodi di pagamento.

SOLLEVARE UN'ECCEZIONE CON UN ASPETTO

Un altro problema che potrebbe verificarsi è che il software, così com'è, non rispetta un requisito fondamentale. Le specifiche indicano che solamente utenti fidelizzati possono acquistare pagando con la tessera fedeltà. Nel caso specifico invece questo non è garantito. Questo è un evidente bug.
L'aspetto che inseriremo per risolvere il problema fa si che al momento dell'esecuzione del metodo *fidelityCardPayement* venga sollevata una *IllegalStateException* nel caso l'utente non sia un cliente fidelizzato.

```
package ioprogrammo;
  public aspect SuperUserAuthenticationAspect
  {
    pointcut authenticationNeeded(Transaction
        testedTransaction): this(testedTransaction) &&
    call(public int Transaction.fidelityCardPayement());

    before(Transaction transaction):
                authenticationNeeded(transaction)
    {
      Customer customer =
                transaction.getCustomer();
      if(!customer.isSuperCustomer())
      {
        throw new IllegalStateException("A normal
            user cannot pay with fidelity card.");
      }
    }
}
```

Il pointcut *authenticationNeeded* è più complesso del precedente. *This(testedTransaction)* cattura l'oggetto *"this"* nel momento in cui si verifica il *pointcut*. L'*and* logico con l'invocazione al metodo *fidelityCardPayement()* fa si che testedTransaction punti alla transazione nella quale si è verificato il *pointcut*. L'*advice before* accetta come parametro proprio la transaction in cui si è verificato il *pointcut*.
Prima di eseguire il metodo l'advice verifica che il cliente intestatario della transazione sia di tipo fidelizzato. In caso negativo l'advice solleva una *IllegalStateException* che interrompe la normale esecuzione.
Ecco l'output prodotto dal programma.

```
[...]
>>starting:Customer
        ioprogrammo.Transaction.getCustomer(
    java.lang.IllegalStateException: A normal user
        cannot pay with fidelity card.
At ioprogrammo.SuperUserAuthenticationAspect
  .ajc$before$ioprogrammo_SuperUserAuthentication
    Aspect$1$20b2d606(SuperUserAuthentication
        Aspect.aj:11)
at ioprogrammo.Transaction.pay(Transaction.java:54)
at ioprogrammo.Customer.run(Customer.java:39)
at ioprogrammo.Customer.main(Customer.java:52)
>>ending:Customer ioprogrammo.Transaction
        .getCustomer() elapsed:10 milliseconds.
[...]
```

CONCLUSIONI

AOP si pone come obiettivo quello di fornire un utile strumento per risolvere una serie di problemi presenti in molti software. Non è certo la panacea di tutti i mali e pare chiaro che un ricorso scellerato ad *AspectJ* sia in grado esclusivamente di peggiorare le doti di manutenibilità di un software.
AOP e *AspectJ* sono sostanzialmente nuovi strumenti nella scatola degli attrezzi dello sviluppatore, ma come sempre, lo sviluppatore deve impegnare il suo cervello per utilizzare armoniosamente tutti gli strumenti a disposizione.

Daniele De Michelis

Combattere gli Hacker

Di sicurezza non se ne parla mai abbastanza. Vi presentiamo le tecniche per criptare messaggi che passano in rete. Utili per esempio per non inviare la vostra password in chiaro attraverso il web

REQUISITI

Conoscenze richieste
Principi di C#

Software
Visual Studio .NET

Impegno

Tempo di realizzazione

La sicurezza delle informazioni e la loro protezione costituiscono un concetto fondamentale, ma spesso erroneamente trascurato a causa dei conseguenti costi di sviluppo. Nelle moderne applicazioni, dove lo scambio di informazioni tra sistemi eterogenei è sempre più frequente, questo concetto diventa la base su cui costruire sistemi sicuri. Pensiamo, ad esempio, ai Web Services in cui l'uso di messaggi in formato XML non criptati o che viaggiano attraverso canali non sicuri, rischia di compromettere severamente il sistema. Talvolta, però, crittografare un messaggio potrebbe non essere sufficiente. Infatti, nulla vieta la modifica del messaggio crittografato, impedendone la corretta lettura da parte del destinatario.
Il .NET Framework offre diverse alternative per risolvere il problema:

- **WSE-Security** *(Web Services Enhancements Security):* se lo scambio dei messaggi avviene attraverso Web Services e sia l'applicazione client sia l'applicazione server sono sviluppate in .NET, è possibile sfruttare le risorse offerte da questa tecnologia;
- **Hashing:** gli algoritmi di hashing permettono di generare un output difficilmente leggibile ed utilizzato come rappresentazione codificata del messaggio originale;
- **Firma digitale:** la firma digitale di file XML, sulla base delle specifiche del W3C.

Algoritmo	.NET Class	Dimensioni del message digest	Produttore
MD5	MD5CryptoServiceProvider	128 bits	*RSA Data Security, Inc*
SHA-1	SHA1CryptoServiceProvider	160 bits	*National Institute of Security and Technology* (NIST), *National Security Agency* (NSA)

Tabbella 1: Gli algoritmi di hashing più diffusi

Il nostro articolo ha lo scopo di illustrare una procedura generale per la risoluzione del problema, eviteremo quindi di parlare di WSE perché troppo legato ad un'implementazione specifica e ci concentreremo invece su hashing e firma digitale.

GLI ALGORITMI DI HASHING

L'*hashing* è una particolare forma di crittografia che consente la generazione di una stringa a dimensione fissa, chiamata *digest* o *hash,* rappresentante il messaggio originale. È importante sottolineare che il *digest* non sostituisce il messaggio originale ma costituisce la sua rappresentazione codificata in una stringa a dimensione fissa. In sostanza blocchi di dati identici generano un identico message *digest,* la modifica anche di un singolo dato comporta, però, la generazione di un output completamente diverso. Generalmente gli output generati sono di lunghezza pari a 128-bit o a 160-bit (**Tabella 1**).
In aggiunta ai due algoritmi di hashing più diffusi, il *NIST* ha proposto tre variazioni dell'algoritmo *SHA-1* tutte supportate dal .NET Framework: *SHA-256, SHA-384* e *SHA-512.*
Gli algoritmi che producono message digest più lunghi sono generalmente più sicuri, appunto perché generano output più difficilmente violabili. Attualmente il più diffuso è SHA-1, che utilizzeremo all'interno del nostro progetto di esempio.

GENERAZIONE DI MESSAGE DIGEST

Creiamo il progetto di esempio seguendo i passi riportati nel tutorial nella pagina a fianco articolo fino ad ottenere un form come quello visualizza-

to in **Figura 1**. Il form consente la generazione di un semplice *message digest* partendo dal dato da analizzare. Aggiungiamo i namespace necessari:

```
using System.Security.Cryptography;
using System.Security.Cryptography.Xml;
using System.Text;
```

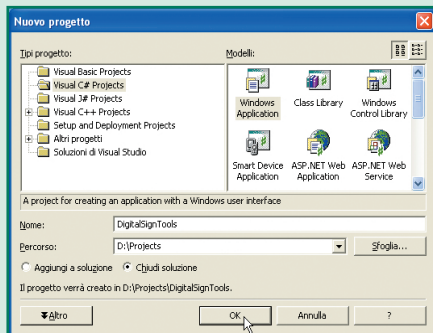
Fig. 1: ***Come si presenta il form della nostra applicazione***

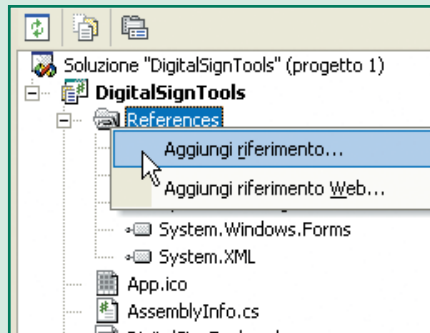
Nel gestore dell'evento click del tasto *Generate*, quindi, riportiamo il seguente codice:

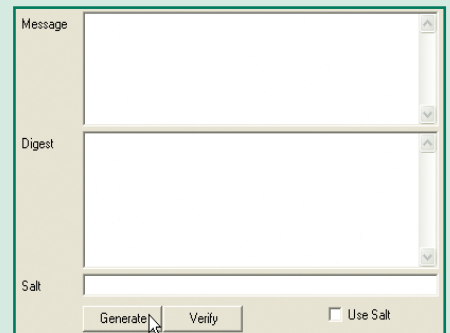
```
byte[] message;
byte[] digest;
SHA1 sha1;
// convertiamo in byte la stringa
message = Encoding.UTF8.GetBytes(
                this.txtMessage.Text);
// istanziamo il provider
sha1 =new SHA1CryptoServiceProvider();
// generiamo il digest
digest = sha1.ComputeHash(message);
```

IL NOSTRO AMBIENTE DI TEST

Sei semplici passi per realizzare praticamente la tecnica descritta nell'articolo

› CREIAMO IL NOSTRO PROGETTO

1 Con Visual Studio .NET creiamo un nuovo progetto Windows C# e lo chiamiamo *"DigitalSignTool"*.

› AGGIUNGIAMO I RIFERIMENTI

2 Aggiungiamo un riferimento all'assembly *System.Security.dll* selezionandolo dalla scheda *".NET"*.

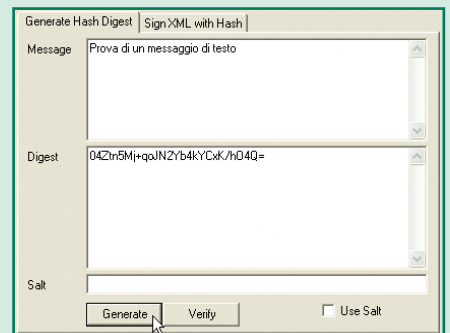
› PREPARIAMO IL FORM

3 Rinominiamo il file *form1.cs* in *mainForm.cs* e realizziamo il form così come riportato in figura.

› INSERIAMO IL CODICE PER L'ENCRYPT

4 Con doppio click sul controllo *bGenerate* aggiungiamo il codice per la codifica.

› INSERIAMO IL CODICE PER LA VERIFICA

5 Il doppio click sul controllo *bVerify* ci permette di aggiungere il codice per la decodifica.

› ESEGUIAMO L'APPLICAZIONE

6 Eseguiamo l'applicazione creata, inseriamo un testo e clicchiamo sul tasto *Encrypt* per eseguirne la crittografia.

```
// convertiamo il digest generato in stringa e lo
                                visualizziamo nel form
this.txtDigest.Text = Convert.ToBase64String(digest);
```

Il codice instanzia il provider managed *SHA1CryptoServiceProvider* e genera il digest in base al messaggio iniziale. Il digest, come vedremo, viaggerà insieme al messaggio originale per consentire al destinatario di verificarne l'integrità.

LA PROTEZIONE DELLE CHIAVI

Durante il processo di creazione della firma digitale vengono utilizzate le chiavi per la crittografia del digest, che devono essere protette per garantire il massimo grado di sicurezza, un procedimento che può avvenire in diversi modi. Uno di questi è sicuramente l'utilizzo delle DPAPI *(Data Protection API)*, che utilizzano le credenziali dell'utente per criptare e decriptare i dati rappresentati, nel nostro caso, dalle chiavi stesse. La memorizzazione del risultato deve essere eseguita su supporti sicuri, come, ad esempio, in cartelle protette o in apposite chiavi nel registro di sistema.

Al momento della ricezione del messaggio, il destinatario provvederà a rigenerare il digest, applicando lo stesso algoritmo al messaggio, e a confrontarlo con il digest ricevuto. Poiché una data stringa genera sempre lo stesso output, se i due digest saranno identici, il messaggio è integro. Se questa condizione non si verifica, possiamo ritenere che il messaggio è stato in qualche modo violato, e la richiesta deve essere respinta. Questo, però, non garantisce l'assoluta sicurezza del sistema. Infatti un messaggio potrebbe essere intercettato, modificato, rigenerato il digest e nuovamente inviato al destinatario. Ovviare a questo inconveniente è semplice quanto ingegnoso. È sufficiente applicare, al momento della creazione del digest, una chiave, altrimenti detta *"salt"* (sale), condivisa tra il mittente ed il destinatario, possibilmente scambiata in momenti differenti dalla trasmissione del messaggio. Vediamo come è possibile ottenere un *hash* accodando al messaggio una chiave random a dimensione fissa:

```
byte[] message;
byte[] salt = new byte[10];
RNGCryptoServiceProvider rng;
SHA1 sha1;
CryptoStream cs;
// convertiamo in byte la stringa contenente il
                                        messaggio
message = System.Text.Encoding.UTF8.GetBytes(
                                this.txtMessage.Text);
// generiamo la chiave utilizzando il Random Number
                                        Generator
rng = new RNGCryptoServiceProvider();
rng.GetBytes(salt);
// istanziamo il provider
sha1 = new SHA1CryptoServiceProvider();
// utilizziamo il CryptoStream per creare il digest
cs =new CryptoStream(Stream.Null, sha1,
                            CryptoStreamMode.Write);
cs.Write(message,0,message.Length);
cs.Write(salt,0,salt.Length);
cs.FlushFinalBlock();
// recuperiamo l'hash creato
this.txtDigest.Text = Convert.ToBase64String(
                                        sha1.Hash);
```

NOTA

Il namespace *System .Security.Cryptography* fornisce la classe astratta *HashAlgorithm*, classe di base, che implementa l'interfaccia *ICryptoTransform*, per tutti gli algoritmi di hashing. Da questa classe eredita la classe astratta *SHA1*, dalla quale ereditano, a loro volta, le classi *SHA1CryptoServiceProvider* e *SHA1Managed*. La prima è un wrapper alle *CryptoAPI*, mentre la seconda è l'implementazione *managed* dell'algoritmo *SHA-1*.

Il codice presentato utilizza il provider *Random Number Generator* per costruire una chiave a dimensione fissa che sarà poi accodata al messaggio nella creazione dell'hash finale. La chiave deve poi essere memorizzata per il successivo utilizzo in fase di verifica, ma non deve viaggiare mai, almeno in chiaro, con il messaggio. Un buon sistema sarebbe quello di criptare la chiave di lettura secondo gli algoritmi standard utilizzabili nel .NET Framework. Eseguendo la verifica dell'hash otteniamo un risultato simile a quello visualizzato in **Figura 2**.

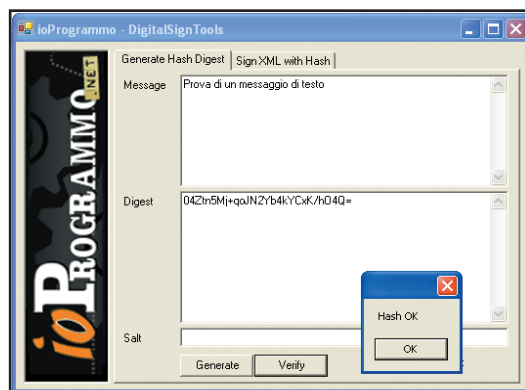



***Fig. 2:* La verifica della firma**

MESSAGE DIGEST DI FILE XML

La tecnica descritta nel paragrafo precedente è generica ed applicabile, quindi, a qualsiasi tipo di messaggio. Ora vedremo, invece, come poter firmare digitalmente un file XML sulla base di quanto appena detto. Il processo per la firma digitale di uno specifico documento avviene calcolando il relativo digest, che viene criptato e spedito insieme al messaggio. In sostanza, il mittente del messaggio crea un digest del documento da firmare. Il digest viene criptato con un algoritmo asimmetrico, utilizzando la chiave privata, che, nella maggior parte dei casi, diventa parte integrante del documento. Per poter essere in grado di leggere il documento, il destinatario, in possesso della chiave pubblica, deve decriptare il digest utilizzando lo stesso algoritmo del mittente, e verificare che il suo hash corrisponda al digest ottenuto. Il processo descritto è rappresentato in **Figura 3**.

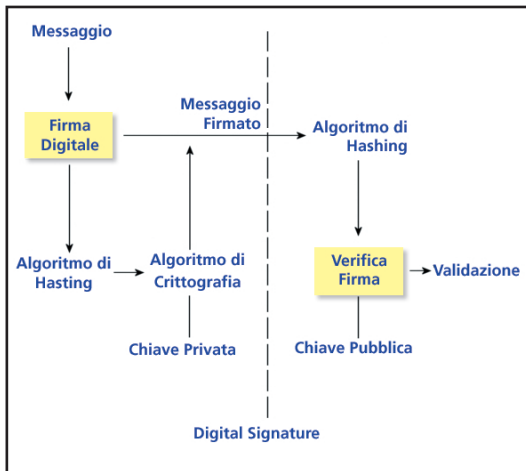



Fig. 3: ***Il ciclo di creazione della firma digitale***

L'utilizzo di algoritmi a chiave asimmetrica consente di ottenere un grado di sicurezza elevato. Inoltre garantisce l'integrità del documento dato che solo chi è in possesso della chiave pubblica può decriptare i messaggi criptati con la corrispondente chiave privata. Il codice di seguito illustrato legge il file *xml* selezionato, aggiunge un nuovo campo al documento contenente il digest criptato del messaggio:

```
byte[] message;
byte[] salt = new byte[10];
stringhash;
RNGCryptoServiceProvider rng =new
                          RNGCryptoServiceProvider();
SHA1 sha1 =new SHA1CryptoServiceProvider();
// carichiamo il file xml selezionato
XmlDocument xml =new XmlDocument();
xml.Load(this.txtHashXMLFile.Text);
// convertiamo in byte il testo da firmare e creiamo
                                            il "salt"
message = Encoding.UTF8.GetBytes(xml.OuterXml);
rng.GetBytes(salt);
// scriviamo nel CryptoStream
CryptoStream cs = new CryptoStream(Stream.Null,
                     sha1, CryptoStreamMode.Write);
cs.Write(message, 0, message.Length);
cs.Write(salt, 0, salt.Length);
cs.FlushFinalBlock();
// criptiamo l'hash generato utilizzando l'algoritmo RSA
RSACryptoServiceProvider rsa = new
                          RSACryptoServiceProvider();
byte[] signedValue = rsa.SignHash(sha1.Hash,
                CryptoConfig.MapNameToOID("SHA1"));
// aggiungiamo il risultato al documento xml
XmlElement xmlHashValue =
                     xml.CreateElement("hashValue");
xmlHashValue.InnerText =
                  Convert.ToBase64String(signedValue);
xml.DocumentElement.AppendChild(xmlHashValue);
```

Il passo successivo consiste nel salvare il documento *xml* risultante o, nel nostro caso, visualizzare il risultato nel form. Per verificare la validità del documento il destinatario deve, dopo aver caricato il file *xml,* leggere e decriptare il digest utilizzando la chiave pubblica in suo possesso. Successivamente viene rimosso il tag *hashValue,* ottenendo così il documento originale del quale viene ricreato il digest. I due digest, infine, vengono confrontati per verificare l'integrità delle informazioni. Il codice completo del progetto è presente nel file sul CD allegato alla rivista. A questo punto è necessaria, però, una piccola riflessione. In alcuni casi due documenti XML, sebbene identici e perfettamente validi dal punto di vista strutturale, possono produrre hash differenti a causa di piccole differenze ignorate dai parser. Per meglio rappresentare questo problema, prendiamo ad esempio le seguenti porzioni XML:

```
<!-- Example 1 -->
<oneElement/>
<!-- Example 2 -->
<oneElement />
<!-- Example 3 -->
<oneElement></oneElement>
```

Come potete notare, i tre elementi sono logicamente validi e tra loro simili. Data l'aggiunta di alcuni caratteri, però, questi tre elementi producono, ovviamente, un differente digest. La risoluzione di questo problema, però, esula dal contenuto del presente articolo, ci limiteremo pertanto a precisare che è fondamentale, onde evitare queste difficoltà, implementare un processo di normalizzazione che garantisca l'utilizzo di un documento xml uguale sia per il mittenteche per il destinatario.

XML CANONICALIZATION

La creazione dei message digest può, in alcuni casi, produrre risultati diversi. Questo perché file *xml* sintatticamente simili possono contenere spazi o caratteri che ne determinano le differenze. Per risolvere questo problema, le applicazioni che implementano l'XML *Digital Segnature* proposta dal *W3C* rispettano le seguenti regole:

1. **I delimitatori di riga sono normalizzati al singolo carattere *#xA,* compresa la sequenza *#xD#xA***
2. **Gli attributi mancanti che hanno valori di default vengono creati con i loro valori di default**
3. **I riferimenti ai caratteri ed alle entità vengono sostituiti con i caratteri o le entità corrispondenti**
4. **I valori degli attributi vengono normalizzati seguendo queste regole**
   a. **Sostituendo le occorrenze dei caratteri #x9, #xA, e #xD con #x20 (spazio) ad eccezione della sequenza #xD#xA, sostituita da un singolo spazio**
   b. **Se l'attributo non è dichiarato per essere un CDATA, viene rimosso ogni interlinea addizionale o spazi sostituendo tutte le occorrenze con un solo spazio.**

È importante notare che i punti 2, 3 e 4b. sono strettamente legati alla presenza di uno schema XML.

SUL WEB

Microsoft Security
http://msdn.microsoft.com/security

W3C Recommendation XML-Signature Syntax and Processing
http://www.w3.org/TR/xmldsig-core

SHA-1 Specification
http://www.itl.nist.gov/fipspubs/fip180-1.htm

Understanding XML Digital Signature (informazioni in lingua inglese)
http://msdn.microsoft.com/library/en-us/dnwebsrv/html/underxmldigsig.asp?frame=true

Il mio blog
http://blogs.ugidotnet.org/fabioc

XML DIGITAL SIGNATURE

Il .NET Framework offre pieno supporto alla standardizzazione offerta dal W3C del processo sopra descritto. L'utilizzo della classe *SignedXml* consente, infatti, di produrre un documento XML che risponde a precise regole. Il punto debole del sistema rappresentato nel paragrafo precedente è costituito dalla sua forte personalizzazione. Si tratta infatti di una struttura creata secondo regole e procedure personali.
L'utilizzo della specifica proposta dal W3C, invece, consente di generare una struttura XML facilmente interpretabile da qualsiasi sistema che ne implementa le caratteristiche. Il codice riportato di seguito sfrutta le classi del namespace *System.Security.Cryptography.Xml* selezionato attraverso il form di input:

```
stringreferenceId = "xmldoc";
RSACryptoServiceProvider rsa =new
                        RSACryptoServiceProvider();
// Preparo il DOMDocument che conterrà il risultato
XmlDocument xmlOutput =new XmlDocument();
XmlDocument xmlInput =new XmlDocument();
xmlInput.Load(this.txtHashXMLFile.Text);
// Costruisco la firma
SignedXml signedXml =new SignedXml();
// Imposto i dati da firmare
System.Security.Cryptography.Xml.DataObject
   dataToSign =new System.Security
                        .Cryptography.Xml.DataObject();
dataToSign.Id = referenceId;
dataToSign.Data = xmlInput.SelectNodes("//*");
signedXml.AddObject(dataToSign);
// imposto l'algoritmo da utilizzare per la firma
signedXml.SigningKey = rsa;
// memorizzo le informazioni sulla chiave di lettura
KeyInfo keyInfo =new KeyInfo();
keyInfo.AddClause(new RSAKeyValue(_rsa));
signedXml.KeyInfo =keyInfo;
// imposto il riferimento ai dati firmati
signedXml.AddReference(new
    Reference(String.Format("#{0}", referenceId)));
// effettuo la firma
signedXml.ComputeSignature();
// Aggiungo la firma al nuovo documento di output
xmlOutput.AppendChild(xmlOutput.ImportNode(
                        signedXml.GetXml(),true));
```

La struttura prodotta include anche informazioni utili per la verifica della validità della firma come il contenuto del tag <*keyInfo*> generato dalla corrispondente classe *KeyInfo* e riportante dati relativi alla chiave utilizzata, che consentono l'identificazione della chiave necessaria per la convalida della firma digitale. Per poter leggere il documento firmato, anche il destinatario utilizza l'oggetto *SignedXml* e, dopo aver recuperato la chiave pubblica in suo possesso, esegue la verifica della firma come di seguito riportato:

```
// Recupero l'intero documento Xml
XmlDocument xml =new XmlDocument();
xml.LoadXml(txtDigestXMLFile.Text);
// Recupero le chiavi per l'utilizzo con l'algoritmo RSA
RSACryptoServiceProvider rsa =new
                        RSACryptoServiceProvider();
rsa.FromXmlString(txtPublicKey.Text);
// Imposto un riferimento all'XmlDocument e
// recupero il nodo contenente la firma
SignedXml signedXml =new SignedXml(xml);
XmlNodeList signature =
            xml.GetElementsByTagName("Signature");
signedXml.LoadXml((XmlElement)signature[0]);
// Verifico la firma
if(signedXml.CheckSignature(rsa))
  MessageBox.Show("Hash OK");
```

CONCLUSIONI

La firma digitale è un argomento di sicuro interesse. Abbiamo voluto applicare questo concetto all'utilizzo con file XML, analizzando anche i problemi che ne derivano. Analizzando le caratteristiche della firma digitale siamo in grado di riprodurne manualmente il comportamento laddove necessario. Ove possibile, infatti, è sempre meglio attenersi agli standard implementando le specifiche stabilite dal W3C.
Buon lavoro.

Fabio Cozzolino

UN ESEMPIO PRATICO

**Supponiamo che dobbiate autenticarvi su un server Web. È un'operazione tipica, che avviene ogni qual volta effettuate un login.
L'idea è che la vostra password non debba viaggiare in chiaro. Vediamo quale potrebbe essere uno schema di funzionamento facendo uso degli algoritmi di criptazione.**

1) **L'utente si connette al Server. Il Server genera un numero random e lo salva in una variabile di sessione.**
2) **L'utente affianca il numero random alla password e genera un hash composto dalla password + il seme.**
3) **La password è il seme criptati con *SHA1* vengono inviati al server con la pressione del tasto *SUBMIT* sulla form.**
4) **Il server effettua una query su un database usando come campo chiave l'username e recupera la password ad esso associato.**
5) **Il server genera un hash del seme salvato nella variabile di sessione più la password recuperata dal database.**
6) **I dati inviati dall'utente vengono confrontati con quelli ottenuti dal server, se i due hash coincidono l'utente è autenticato.**

Si tratta di un processo che può ancora essere complicato, tuttavia per i nostri scopi è sicuramente il primo passo per iniziare.

BIBLIOGRAFIA

- **PRACTICAL CRYPTOGRAPHY** ***Niels Ferguson e Bruce Shneier*** **(Wiley Publishing) ISBN 0-471-22357-3 (2003)**
- **.NET FRAMEWORK SECURITY** ***Brian A. LaMacchia, Sebastian Lange, Matthew Lyons, Rudi Martin, Kevin T. Price*** **(Addison Wesley) ISBN 0-672-32184-X (2002)**
- **CRYPTOGRAPHY AND DATA SECURITY** ***Dorothy Denning*** **(Addison Wesley)**

"Riflettendo" sul codice .NET

Introduciamo uno dei concetti più importanti della programmazione .NET: la Reflection. Implementeremo un assembly/class browser indipendente dalla disponibilità del codice

Spesso capita di riflettere sul codice che scriviamo, o che qualcun altro ha scritto prima di noi, e di cercare di capire cosa fa e come funziona. Proprio "riflettendo" sul codice, mi è venuto in mente che sarebbe utile automatizzare questo processo, e farmi aiutare dalle classi del framework .NET per scoprire come è strutturata una qualsiasi classe, sia essa fornita dalla *Base Class Library*, oppure una che abbiamo scritto qualche tempo fa e non ci ricordiamo più come è fatta, o peggio, abbiamo smarrito il suo codice sorgente e ci servirebbe sapere qual era la firma di un particolare metodo.
Pensate poi se vi trovate da un cliente, a dover sviluppare qualche applicazione .NET senza avere a disposizione né Visual Studio né altri IDE, né la documentazione, e non vi ricordate quali erano quei parametri di quel metodo di quella classe, potete tirarvi fuori dai guai utilizzando la *Reflection*, sempre se vi ricordate i metodi e le classi del namespace *System.Reflection*.

Fig. 1: Uno screenshot dell'applicazione che creeremo

REQUISITI

Conoscenze richieste
Conoscenze medie del framework .NET e C#

Software
.NET Framework SDK

Impegno

Tempo di realizzazione

In questo articolo vedremo come creare una semplice, ma non troppo, applicazione per l'esplorazione dei tipi, e di cui potete ammirare una schermata finale in **Figura 1**.

COS'È LA REFLECTION

Il concetto di reflection permette di esplorare il contenuto di qualsiasi assembly o modulo .NET a runtime, e scoprire ad esempio quali siano i tipi contenuti in esso ed i relativi campi, le proprietà, gli eventi, e quant'altro una classe espone pubblicamente e non.
Naturalmente lo scopo e le possibilità della *Reflection* non sono solo questi, ed i suoi possibili utilizzi sono quindi molteplici.
Ad esempio è possibile scrivere applicazioni estensibili tramite *add-on*, oppure, come fa ad esempio visual studio, visualizzare in una finestra le proprietà di un oggetto selezionato nel designer, o ancora visualizzare l'elenco dei metodi che una classe possiede per creare dei sistemi di completamento automatico del codice.
Per mezzo delle classi contenute nel namespace *System.Reflection* e della classe *System.Type*, realizzaremo un semplice ma efficace tool per ricavare la struttura di un tipo, dalla classe madre, alle eventuali interfacce implementate, passando per ogni suo membro.

ASSEMBLY, MODULI E TIPI

Le classi chiave per ricavare informazioni su assembly e moduli, sono le classi *Assembly* e *Module* del namespace *System.Reflection*, mentre un tipo sarà naturalmente rappresentato dalla classe *System.Type*.

Fig. 2: *La gerarchia delle classi System.Reflection*

Altre classi ci permetteranno invece di ricavare informazioni a runtime su ogni membro di un tipo.
La gerarchia delle classi fondamentali facenti parte del namespace *System.Reflection* è riassunta nella **Figura 2**.
Supponiamo di avere un assembly di cui vogliamo ricavare ed esplorare il contenuto, allo scopo possiamo crearne uno nostro compilando un qualsiasi sorgente .NET, oppure utilizzare un assembly standard del *framework .NET,* ad esempio *System.dll.*
Grazie al metodo statico *LoadFrom* carichiamo l'assembly:

```
string pathAsm=@"c:\WINDOWS\Microsoft.NET
                        \Framework\v1.1.4322\System.dll";
Assembly asm = Assembly.LoadFrom (pathAsm);
```

È anche possibile ricavare informazioni sull'assembly in esecuzione, mediante il metodo statico *GetExecutingAssembly:*

```
asm=Assembly.GetExecutingAssembly();
```

La classe *Assembly* fornisce un numeroso elenco di metodi per trovare informazioni sul contenuto di un eseguibile o di una libreria. Per ricavare i tipi che sono definiti nell'assembly basta ad esempio utilizzare il metodo *GetTypes,* che retituisce un array di oggetti *Type:*

```
Type[] types=asm.GetTypes();
foreach(Type t in types)
{
   Console.WriteLine(t.FullName);
}
```

IL TIPO FONDAMENTALE

Al centro del concetto di *Reflection* sta la classe *Type*. Essa rappresenta un tipo qualunque del *Common Type System* di .NET e quindi permette di ricavare diverse informazioni.
La classe *Type* infatti possiede diverse proprietà booleane, che indicano ad esempio se un tipo è astratto, se si tratta di un'interfaccia, se è una classe sealed, oppure ancora proprietà che consentono di ottenere i modificatori di accesso della classe.
Il metodo seguente, dato un oggetto qualunque, ne ricava il tipo, e restituisce l 'intestazione di classe, facendo uso delle proprietà della classe *Type* e del metodo *GetInterfaces* per ricavarne le interfacce implementate.

```
public static void PrintTypeInfo(Type type)
{
   Console.WriteLine("\n-----------------------------");
   Console.WriteLine("--------PrintTypeInfo--------\n");
   string str="";

   try
   {
      if(type.IsPublic)
         str+="public ";
      if(type.IsNotPublic)
         str+="internal ";
      if(type.IsSealed)
         str+="sealed ";
      if(type.IsAbstract)
         str+="abstract ";

      if(type.IsClass)
         str+="class ";
      else if(type.IsInterface)
         str+="interface ";
      else if(type.IsEnum)
         str+="enum ";
      else if(type.IsValueType)
         str+="struct ";

         str+=type.Name;
         str+=":"+type.BaseType.FullName;
```

I TUOI APPUNTI

UTILIZZO DEI BINDINGFLAGS

I membri dell'enumerazione *BindingFlags* consentono di specificare come il meccanismo debba condurre la ricerca dei membri di un determinato tipo. Poiché l'enumerazione ha un attributo *FlagsAttribute* è possibile combinare in *OR bitwise* i suoi membri. Per ottenere risultati è necessario specificare almeno uno fra *Instance* o *Static,* assieme ad almeno una fra *Public* e *NonPublic.* Inoltre il membro *DeclaredOnly,* se presente nella combinazione *OR,* consentirà di ottenere solo i metodi dichiarati direttamente in un tipo, e non quelli ereditati.

```
        foreach(Type interf in type.GetInterfaces())
        {
            str+=","+interf.FullName;
        }
    }
    catch(Exception ex)
    {
        Console.WriteLine(ex.Message+"\n"+ex.StackTrace);
    }

    Console.WriteLine(str);
}
```

Scriviamo delle classi di test, dall'utilità nulla in altri casi, ma che ci serviranno a testare il metodo precedente.

```
interface UnInterfaccia
{
}

public class UnaClasse: UnInterfaccia
{
}

internal abstract class UnaClasseDerivata: UnaClasse
{
}
```

Invocando ora il metodo *GetTypeInfo* nella maniera seguente otterremo proprio le intestazioni dei tipi che abbiamo appena scritto:

```
static void Main()
{
    PrintTypeInfo (typeof(UnInterfaccia));
    PrintTypeInfo(new UnaClasse ().GetType());
    PrintTypeInfo(Type.GetType("UnaClasseDerivata"));
}
```

Nel metodo *Main* potete notare come sia possibile ricavare un oggetto *Type* in diverse maniere, ad esempio attraverso l'uso di *typeof*, oppure invocando il metodo *GetType* su un'istanza di una classe, o ancora utilizzando il metodo statico *Type.GetType* ed il nome del tipo.

ENUMERARE I MEMBRI DI UNA CLASSE

Nel paragrafo precedente abbiamo utilizzato la classe *Type* per ricavare l'intestazione di una qualunque classe.
Adesso vedremo come ricavarne anche i membri, siano essi campi, metodi, proprietà, eventi, ed esaminarli utilizzando le classi del namespace *System.Reflection*. La classe *type* infatti fornisce un metodo generale *GetMembers* adatto a tale scopo. Esso però permette di ricavare solo i membri che costituiscono l'interfaccia pubblica di una classe. Inoltre diversi metodi sono dedicati ad un particolare membro, ad esempio *GetFields*, *GetProperties*, *GetMethods*, con diversi overload per specificare ad esempio la visibilità degli elementi da ricavare.
Il metodo *GetMembers* restituisce un'array di oggetti *MemberInfo*, ognuno dei quali rappresenta e contiene informazioni su un membro della classe. Per sapere con che tipo di membro stiamo avendo a che fare possiamo esaminare la proprietà *MemberType*, mentre la proprietà *Name* ci restituisce il nome. Ad esempio, con il seguente ciclo enumeriamo nome e tipo dei membri pubblici di un tipo:

```
foreach(MemberInfo mi in t.GetMembers())
{
    Console.WriteLine("{0}
                        {1}",mi.MemberType,mi.Name);
}
```

ENUMERARE CAMPI METODI E PROPRIETÀ

I vari metodi *GetXXXs* della classe *Type* restituiscono degli array di tipo *XXXInfo*, dove *XXX* è una delle classi che abbiamo visto nella gerarchia rappresentata in **Figura 2**, in particolare le classi derivate a partire dalla *MemberInfo*. Ad esempio il metodo *GetFields* restituirà un array di oggetti *FieldInfo*.
I metodi suddetti, invocati senza alcun parametro, restituiranno però solo i relativi oggetti dichiarati *public* nella classe. Se invece vogliamo specificare dei parametri di visibilità, utilizzeremo un argomento di classe *Binding Flags*, che è un'enumerazione i cui valori possono essere combinati tramite l'operatore di *OR bitwise* per specificarne più di uno alla volta. Ad esempio il metodo *GetMethods* senza parametri è equivalente alla seguente chiamata:

```
MethodInfo[] metodiPubblici=
                GetMethods(BindingFlags.Instance |
            BindingFlags.Static | BindingFlags.Public);
```

Con la combinazione in *OR* dei tre valori di *BindingFlags*, si specifica di voler ricavare i metodi pubblici,sia statici sia di istanza.
Analogamente potremo ricavare metodi privati, protected o quel che vogliamo. Per chia-

NOTA

METADATI

Il codice compilato in assembly .NET, viene strutturato in tabelle di metadati. Ad esempio viene creata una tabella dei tipi, una tabella dei campi, una dei metodi, e così via. Le classi del namespace System.Reflection effetttuano un parsing di tali tabelle, fornendo quindi allo sviluppatore un modello ad oggetti al di sopra dei nudi e crudi metadati.

rimenti sull'utilizzo dell'enumerazione *BindingFlags*, fate riferimento alla tabella relativa contenuta in queste pagine. Quindi se volessimo ricavare i campi d'istanza di una classe, non ereditati da classi superiori, dovremo semplicemente invocare il metodo *GetFields* così:

```
FieldInfo[] fields=type.GetFields(
        BindingFlags.Instance | BindingFlags.Public |
   BindingFlags.NonPublic | BindingFlags.DeclaredOnly);
```

Lo stesso ragionamento si applica naturalmente ai metodi *GetProperties*, *GetConstructors*, e così via, come vedremo nel paragrafo seguente.

UN VISUALIZZATORE DI CLASSI

Utilizzeremo adesso i concetti esposti finora in maniera pratica, creando una sorta di *Type Viewer*, cioè un visualizzatore della struttura di una qualsiasi classe contenuta in un qualsiasi assembly .NET.
Iniziamo dunque dal caricamento degli assembly. Nella Form principale dell'applicazione utilizzeremo un controllo *TreeView* per visualizzare gli assembly caricati, e le classi in essi contenute, suddivise naturalmente per namespace.
Per semplicità nel nostro progetto leggeremo la lista degli assembly da un file di testo, in cui essi saranno stati inseriti preventivamente, uno per riga, con il percorso completo da cui caricarli. Un estratto del file assemblies.txt potrebbe essere questo:

```
C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322
                                \mscorlib.dll
C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322
                        \System.Windows.Forms.dll
```

Il metodo *LoadPredefAssemblies* legge il contenuto del file e crea un array di stringhe contenente i percorsi degli assembly:

```
protected void LoadPredefAssemblies()
{
  try
  {
    StreamReader sr=File.OpenText(strAsmFile);
    string line;
    while((line=sr.ReadLine())!=null)
    {
      if(File.Exists(line))
        assemblies.Add(line);
```

```
      }
  }
  catch
  {
    MessageBox.Show("Impossibile aprire il file degli
                                    assembly");
  }
}
```

Inoltre daremo anche la possibilità di aprire un qualsiasi assembly *exe* o *dll* attraverso una classica *OpenFileDialog*. Tralasciando il codice per creare la Form ed i vari controlli, mostriamo i passi fondamentali per caricare le classi presenti in un dato assembly.
Con il seguente codice, viene caricato un assembly, e creato un nodo dell'albero che fungerà da root per ogni namespace in esso contenuto, ed a sua volta per ogni classe facente parte del namespace:

```
Assembly asm;
TreeView tvwTypes=new TreeView:
TreeNode node;
…
asm=Assembly.LoadFrom(pathAssembly);
node=new TreeNode(asm.GetName().Name,0,0);
node.Tag=asm;
tvwTypes.Nodes.Add(node);
AddAssemblyTypes(node);
```

Il metodo *AddAssemblyTypes*, dato il nodo passato in ingresso, ricava i tipi contenuti in un assembly, suddividendoli per namespace. Per far ciò verifica se il node che rappresenta il namespace già esiste, in caso contrario lo aggiunge all'albero.

```
Type[] types=asm.GetTypes();
TreeNode nodeNS,nodeType=null;
foreach(Type t in types)
{
  nodeNS=FindNamespaceNode(
                    asmnode,t.Namespace);
  if(nodeNS==null && t.Namespace!=null)
  {
    nodeNS=new TreeNode(t.Namespace,1,1);
    nodeNS.Tag=t.Namespace;
    asmnode.Nodes.Add(nodeNS);
  }
  if(t.Namespace!=null)
  {
    nodeType=null;
    if(t.IsClass)
    {
      nodeType=new TreeNode(t.Name,2,2);
    }
    else if(t.IsInterface)
```

CODICE "IL"

La *Reflection* non permette di ricavare il codice in linguaggio intermedio *IL*, in quanto si basa solo sulle tabelle di metadati. Per creare applicazioni come *ILDasm* o i vari decompilatori commerciali, bisogna effettuare quindi direttamente il parsing dei file, byte per byte, studiando il formato standardizzato dall'*ECMA*, onde ricavare il corpo dei metodi.

```
    {
        nodeType=new TreeNode(t.Name,3,3);
    }
    else if(t.IsEnum)
    {
        nodeType=new TreeNode(t.Name,4,4);
    }
    else if(t.IsValueType)
    {
        nodeType=new TreeNode(t.Name,4,4);
    }
    if(nodeType!=null)
    {
        nodeType.Tag=t;
        nodeNS.Nodes.Add(nodeType);
    }
  }
}
```

Come potete notare dalla **Figura 3**, in questa maniera l'albero verrà popolato di nodi rappresentanti assembly, namespace, classi, interfacce, strutture ed enumerazioni. A questo punto gestiremo il doppio click su uno dei nodi rappresentanti un tipo, per ricavare tutti i suoi membri, e le relative caratteristiche.

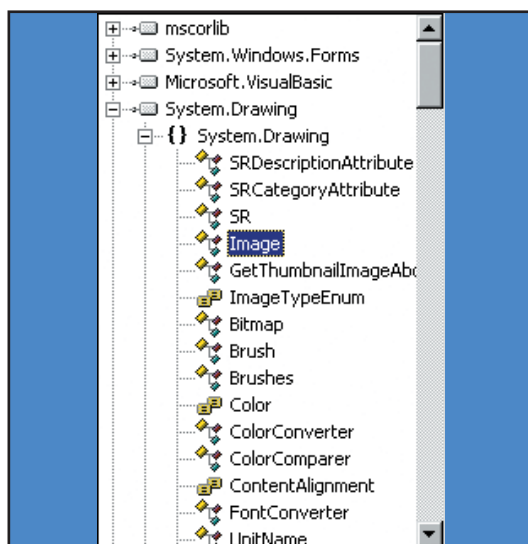



Fig. 3: *Un albero dei tipi contenuti negli assembly*

IL CONTROLLO TYPEEXPLORER

Per ricavare e mostrare i membri di un tipo selezionato dall'albero, ho utilizzato uno user control ad hoc. Per chi fosse interessato alla sua realizzazione può trovare il codice completo nel file *TypeExplorerCtl.cs.* Ciò che invece è più legato al nostro articolo, è il ricavare le informazioni sui membri di un dato tipo.
Per ottenere queste informazioni viene invocato il metodo *ShowClassInfo*, passandogli un parametro di classe *Type*, che rappresenta il tipo da esplorare.

```
private void ShowClassInfo(Type t)
{
  txtAssembly.Text=t.Assembly.FullName;
  txtNamespace.Text=t.Namespace;
  txtClassname.Text=t.Name;
  if(t.BaseType!=null)
     txtBasetype.Text=t.BaseType.FullName;
  if(t.IsAbstract)
     chkAbstract.Checked=true;
  if(t.IsSealed)
     chkSealed.Checked=true;
  if(t.IsInterface)
     chkInterface.Checked=true;
  if(t.IsValueType)
     chkValue.Checked=true;
  if(t.IsEnum)
     chkEnum.Checked=true;
  if(t.IsPrimitive)
     chkPrimitive.Checked=true;
  Type[] interfaces=t.GetInterfaces();
  foreach(Type itf in interfaces)
  {
      lstInterfaces.Items.Add(itf.FullName);
  }

}
```

Praticamente con le sole proprietà della classe *Type*, ricaviamo ogni sorta di attributo su un tipo. Il metodo *GetInterfaces* invece ci restituisce le interfacce che implementa.
Il risultato è mostrato nella parte alta del controllo, come mostrato in **Figura 4**.

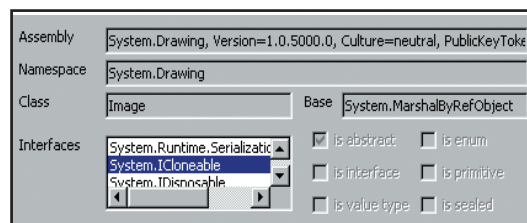



Fig. 4: *Gli attributi di una classe*

Non resta adesso che ricavare i campi, le proprietà, ed i metodi, costruttori inclusi, della classe. Inoltre selezionando ognuno di essi, mostreremo anche informazioni specifiche, ad esempio per una proprietà mostreremo i suoi modificatori di accesso e se si tratta di una proprietà get oppure set, o entrambi.
I tre metodi *ShowClassFields*, *ShowClassProperties* e *ShowClassMethods*, sono abbastanza simili, mostriamo dunque solo il corpo del primo rimandando al codice allegato even-

• **APPLIED MICROSOFT .NET PROGRAMMING** ***Richter*** **(Microsoft Press)**

• **C# AND .NET PLATFORM** ***Troelsen*** **(APress)**

tuali dubbi sugli altri:

```
FieldInfo[] fields=t.GetFields(
        BindingFlags.DeclaredOnly|BindingFlags.Public|
          BindingFlags.NonPublic|BindingFlags.Instance
                                   |BindingFlags.Static);
DataTable dt=new DataTable("Fields");
dt.Columns.Add("FieldName",typeof(string));
dt.Columns.Add("FieldInfo",typeof(FieldInfo));
DataRow row;
foreach(FieldInfo field in fields)
{
   row=dt.NewRow();
   row[0]=field.Name;
   row[1]=field;
   dt.Rows.Add(row);
}
lstFields.DisplayMember="FieldName";
lstFields.ValueMember="FieldInfo";
lstFields.DataSource=dt;
if(lstFields.Items.Count>0)
   lstFields.SelectedIndex=0;
```

La *ListBox* dei campi viene riempita impostando la sua proprietà con una *DataTable*, le cui colonne contengono rispettivamente il nome del campo ed un oggetto *FieldInfo*. In questa maniera, utilizzando la prima colonna come *DisplayMember* e la seconda come *ValueMember* della ListBox, sarà immediato mostrare i nomi dei campi all'utente, mentre quando cliccheremo su uno di essi potremo ottenere direttamente l'oggetto *FieldInfo* ad esso associato tramite la proprietà *SelectedValue*. Il gestore del click infatti effettua queste operazioni:

```
FieldInfo fi=lstFields.SelectedValue as FieldInfo;
if(fi!=null)
ShowFieldAttributes(fi);
```

Il metodo *ShowFieldAttributes* si occupa invece di ricavare tutti gli attributi di un *FieldInfo*, in questa maniera:

```
txtFieldType.Text=field.FieldType.FullName;
if(field.IsPrivate)
   chkPrivateField.Checked=true;
if(field.IsFamily)
   chkProtectedField.Checked=true;
if(field.IsAssembly)
   chkInternalField.Checked=true;
if(field.IsPublic)
   chkPublicField.Checked=true;
if(field.IsFamilyOrAssembly)
{  chkInternalField.Checked=true;
   chkProtectedField.Checked=true;
}
```

I metodi che mostrano attributi su proprietà e metodi sono praticamente uguali, vediamo quindi solo come, per un metodo, possiamo ricavare i suoi eventuali parametri.
Anche un parametro è naturalmente rappresentato da un oggetto, di classe *ParameterInfo*. Il metodo *GetParameters* della classe *MethodInfo* restituisce tutti i parametri del metodo:

```
ParameterInfo[] parameters=
                              method.GetParameters();
foreach(ParameterInfo par in parameters)
{
    lstMethodParams.Items.Add(
          par.ParameterType.FullName+" "+par.Name);
}
```

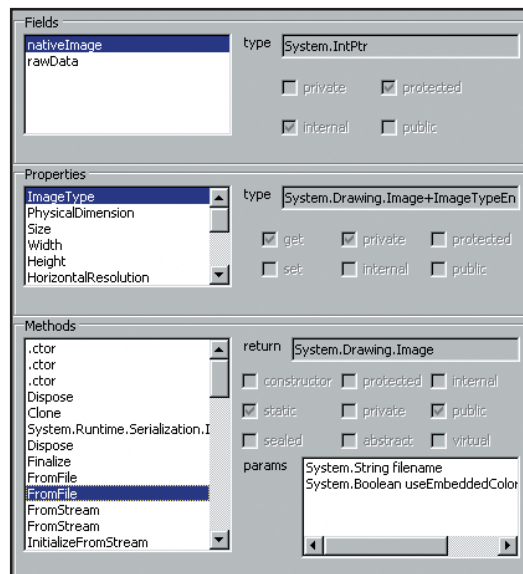



Fig. 5: I membri della classe Image

In **Figura 5** potete osservare come il controllo *typeExplorerCtl* mostra i membri della classe *System.Drawing.Image*.

IL PROGETTO È PRONTO

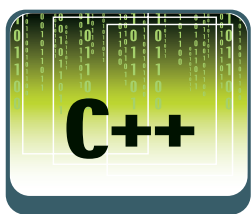
A questo punto l'applicazione è terminata, e sebbene sia servita come esempio per questo articolo sulla *reflection*, portrebbe anche avere la sua utilità pratica, qualora non si avesse a disposizione l'SDK di .NET, Visual Studio con il suo *object viewer*, o qualche disassemblatore.
La reflection naturalmente non si limita a questo, vi sono anche aspetti più avanzati, come l'invocazione dinamica dei metodi, o la creazione di tipi a runtime, ne parleremo in futuro.

Antonio Pelleriti

CONTATTA L'AUTORE

Potete contattare l'autore per suggerimenti, critiche o chiarimenti all'indirizzo e-mail antonio.pelleriti@ioprogrammo.it, **oppure sul sito** www.dotnetarchitects.it **e ovviamente sul forum di ioProgrammo** http://forum.ioprogrammo.it

Parallax Mapping illusione o realtà

Una tecnica piuttosto sofisticata che consente di simulare l'effetto profondità su superfici piane. Riproduce un comportamento tipico dell'occhio umano creando copie quasi perfette

Qualche tempo fa il settore della grafica 3D real-time era orientato verso lo sviluppo di tecniche che consentivano di ottenere il maggior numero possibile di poligoni disegnati a schermo. Oggigiorno la partita si gioca anche su un altro fronte, quello degli "effetti visivi", cioè tecniche che rendono possibile "ingannare" l'occhio dell'utente, facendogli percepire un livello di dettaglio geometrico che in realtà non esiste. Tra queste tecniche, di cui abbiamo parlato su queste stesse pagine (*Pixel* e *Vertex Shaders, Normal Mapping*) figura anche quella del *Parallax Mapping* (PM), che certamente è tra quelle di maggiore impatto visivo.

SPIAZZAMENTO E PARALLASSE

Il PM è una tecnica che permette di simulare una superficie molto complessa geometricamente, attraverso la deformazione della texture che su questa superficie è applicata. In altre parole si disegna su una superficie piatta una immagine in modo tale che sia esattamente uguale a quella che si percepirebbe se la superficie fosse irregolare. Per rendersi conto di cosa

Fig. 1: Su una superficie piana viene applicata una texture simulando la percezione della profondità

si sta parlando si può guardare la **Figura 1**, che riprende un'opera di un geniale artista di strada. La deformazione tiene conto dell'errore di parallasse indotto dall'osservazione di una superficie irregolare da una certa angolazione.
Quando una texture rappresentante una superficie irregolare è applicata su di un semplice poligono, essa appare appiattita. Quello che accade è intuibile guardando la **Figura 2**.

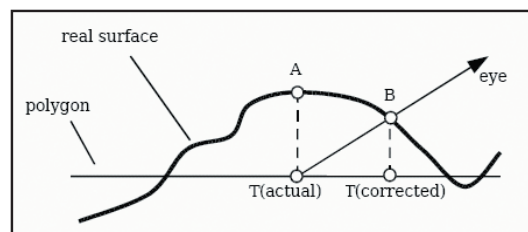



Fig. 2: In condizioni normali il punto viene proiettato sulla superficie non tenendo conto dell'angolo di proiezione dell'occhio

La linea orizzontale è il poligono su cui giace la texture, mentre la linea ondulata è quella che, idealmente, dovrebbe essere la superficie irregolare. Con una normale tecnica di *texture mapping* guardando verso il punto *T* troveremo il pixel relativo al punto *A* della superficie reale.
Questo punto è ben diverso da quello che in realtà dovremmo vedere, e che è rappresentato nel disegno dalla lettera *B*. Questa differenza tra pixel effettivamente percepito e pixel che dovrebbe essere percepito è causa dell'appiattimento di cui si parlava: la superficie irregolare è "spalmata" su una superficie completamente piatta, e si vede!
Il PM tenta di porre rimedio a questa differenza percettiva basandosi su un principio: più la superficie irregolare sarà "alta" sul poligono, più sarà lontana dall'occhio dell'osservatore e dovrà quindi essere disegnata con un certo spiazzamento *(offset)*.

Fig. 3: ***Una rappresentazione corretta con il PA potrebbe essere quella in figura***

Questo spiazzamento è rappresentato in **Figura 3**. Minore, invece, è la distanza tra superficie piatta e superficie irregolare e minore sarà lo spiazzamento per il pixel interessato. Si può notare in figura come l'utilizzo dell'offset sia una approssimazione di quello che dovrebbe essere il reale punto percepito.
Tutto ciò è molto ragionevole, in quanto consente di ottenere "quasi" il risultato "perfetto", utilizzan do una piccola percentuale delle risorse di calcolo altrimenti necessarie. Lo spiazzamento può essere calcolato, nell'implementazione dell'algoritmo, tramite una semplice *height map* (mappa di altezze) che altro non è che una texture particolare, per la quale ciascun pixel non rappresenta un colore ma una altezza dalla superficie, generalmente compresa tra i valori 0.0 e 1.0.

PARALLAX MAPPING E IRRLICHT

Implementare da zero un algoritmo di PM, per quanto semplice, richiederebbe un notevole sforzo, sia progettuale che di programmazione. Considerando poi che esistono numerose varianti all'algoritmo originale, si rischierebbe di non arrivare in tempi ragionevoli ad apprezzare un qualche effetto "tangibile". Per questo utilizzeremo nel resto dell'articolo l'implementazione del PM offerta da IrrLicht. IrrLicht è un framework di sviluppo di applicazioni in real-time 3D gratuito, open e molto potente, che vanta una enorme comunità di sviluppatori (e un seguito entusiasta tra i lettori di ioProgrammo!). Dalla versione 0.10 IrrLicht supporta il PM in maniera nativa. La cosa da fare immediatamente quindi è scaricare l'ultima versione disponibile dell'SDK al sito *http://irrlicht.sourceforge .net/* e cominciare a "smanettare" col codice che proponiamo di seguito.
L'effetto finale del programma completo che realizzeremo sarà quello di una stanza dalle pareti composte da spesse mura di mattoni di pietra. I mattoni sembreranno davvero "solidi" e dotati di una propria geometria, nonostante siano in realtà delle semplici texture con height map, passate sotto la "cura di bellezza" del PM.

IL CODICE

Il codice che proponiamo è modellato sull'esempio *PerPixelLighting* e pertanto va eseguito da una cartella allo stesso livello di quella degli esempi in IrrLicht.
Il corpo principale del programma comincia così:

```
// Corpo principale del programma
int main()
{
    // Creiamo il device
    IrrlichtDevice* device = createDevice(
                                    video::EDT_DIRECTX9,
      core::dimension2d<s32>(640, 480));
    if (device == 0)
        return 1;

    // Otteniamo i puntatori che ci interessano
    video::IVideoDriver* driver =
                                device->getVideoDriver();
    scene::ISceneManager* smgr =
                                device->getSceneManager();
    driver->setTextureCreationFlag(
                    video::ETCF_ALWAYS_32_BIT, true);
```

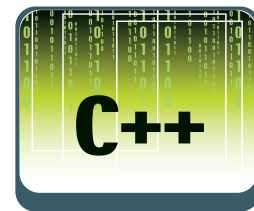
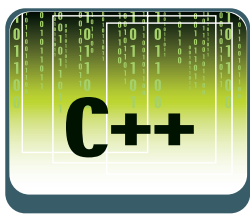
Abbiamo istanziato un device che utilizza le DirectX 9 e disegna in una finestra di dimensioni 640x480 pixel. Abbiamo ottenuto i puntatori agli oggetti che verranno utilizzati in seguito (*VideoDriver* e *SceneManager*) e abbiamo forzato la creazione di texture a 32 bit.
Un altro passo utile è quello di creare una videocamera manipolabile coi tasti freccia e col mouse...

```
    // Aggiungiamo una videocamera di tipo FPS
    scene::ICameraSceneNode* camera =
                        smgr->addCameraSceneNodeFPS(
                                        0,100.0f,300.0f);
    camera->setPosition(core::vector3df(
                                        -200,200,-200));
```

nonché quello di sbarazzarci dell'antiestetico cursore del mouse:

```
    // Diabilitiamo il cursore del mouse
    device->getCursorControl()->setVisible(false);
```

Cominciamo a fare sul serio caricando la mesh della stanza di mattoni che andremo a rappresentare. Questa operazione è completamente automatizzata in IrrLicht ed è possibile ottenere

I TUOI APPUNTI

la mesh semplicemente chiamando la funzione *getMesh()* con il nome di un file *.3ds*.

```
// Carichiamo la mesh della stanza dal suo
                                    modello 3ds
scene::IAnimatedMesh* roomMesh =
                               smgr->getMesh(
    "../../media/room.3ds");
scene::ISceneNode* room = 0;
```

Se l'operazione è andata a buon fine, carichiamo le mappe che ci interessano, e precisamente la texture vera e propria *(rockwall.bmp)*, che determina il colore dei mattoni, e la height map *(rockwall_height.bmp)*, che stabilisce l'altezza della texture dalla superficie piatta cui è applicata.

```
if (roomMesh)
{
    smgr->getMeshManipulator()->
                       makePlanarTextureMapping(
          roomMesh->getMesh(0), 0.003f);
    video::ITexture* colorMap = driver->
      getTexture("../../media/rockwall.bmp");

    // Carichiamo la height map...
    video::ITexture* normalMap = driver->
      getTexture(
              "../../media/rockwall_height.bmp");
```

La height map e la mesh caricate non forniscono tuttavia informazioni essenziali per l'utilizzo della tecnica che stiamo descrivendo (ad esempio normali, tangenti ecc.). Potremmo generare questi dati attraverso un apposito editor 3D, oppure lasciare che sia IrrLicht a effettuare l'operazione. Optiamo, per semplicità, per questa seconda ipotesi:

```
    // ... e creiamo una normal map utilizzando
                          l'apposita funzione!
    driver->makeNormalMapTexture(
                          normalMap, 9.0f);

    // Creiamo una mesh con tangenti dalla mesh
    // della stanza precedentemente caricata
    scene::IMesh* tangentMesh = smgr->
                        getMeshManipulator()->
      createMeshWithTangents(
                        roomMesh->getMesh(0));
```

Una volta creata la mesh e la mappa con tutte le informazioni del caso, impostiamo la normal map in modo che sia associata alla mesh caricata, e aggiungiamo le mesh stessa a un nodo della scena, in modo che IrrLicht possa disegnarla a schermo successivamente.

```
    // Aggiungiamo la mesh così creata alla scena
    room = smgr->addMeshSceneNode(
                              tangentMesh);

    // Impostiamo le mappe caricate e l'EmissiveColor
    room->setMaterialTexture(0,    colorMap);
    room->setMaterialTexture(1,    normalMap);
    room->getMaterial(0).EmissiveColor.set(
                                    0,0,0,0);

    // Il material deve essere quello del Parallax
                                     Mapping
    room->setMaterialType(
              video::EMT_PARALLAX_MAP_SOLID);
    room->getMaterial(0).MaterialTypeParam =
                                      0.035f;
    // height dell'effetto parallasse

    // drop della mesh
    tangentMesh->drop();
}
```

Siccome gli effetti di questo tipo sono valorizzati dalla presenza di luci dinamiche, aggiungiamo una fonte luminosa che si muove in cerchio all'interno della stanza creata. Per il movimento utilizziamo un *FlyCircleAnimator*, un *Animator* di IrrLicht che muove un nodo della scena se-

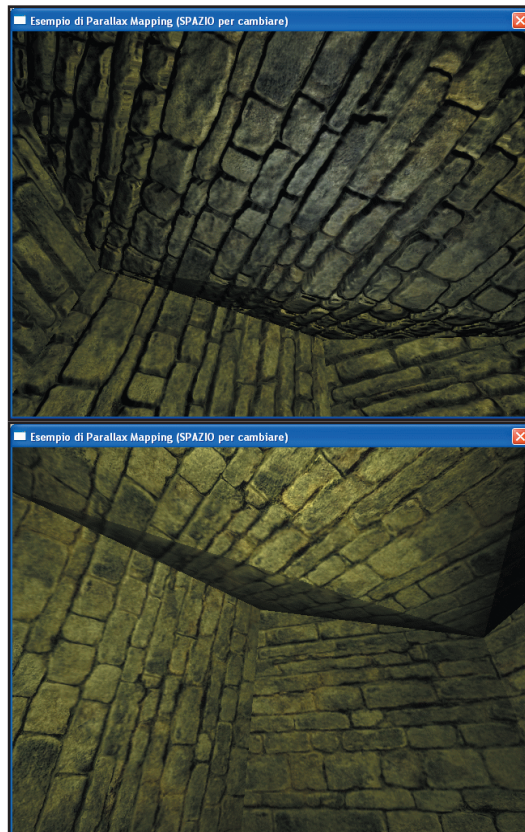



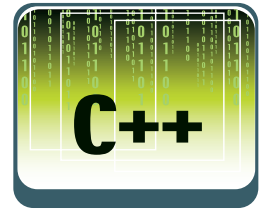
Fig. 4: ***Si noti la differenza fra l'immagine con gli effetti di luce applicati e l'immagine priva di effetti di luce***

condo una traiettoria circolare, con una certa velocità.

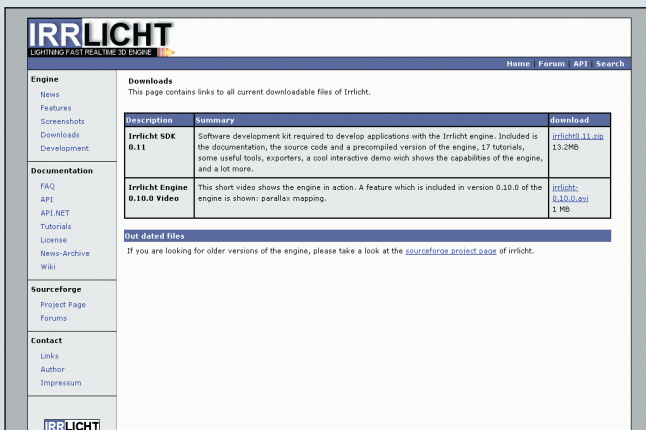
```
// Aggiungiamo una luce e la animiamo in cerchio
scene::ILightSceneNode* light1 =
    smgr->addLightSceneNode(
                    0, core::vector3df(0,0,0),
    video::SColorf(1.0f, 1.0f, 0.5f, 0.0f), 200.0f);

scene::ISceneNodeAnimator* anim =
    smgr->createFlyCircleAnimator
        (core::vector3df(50,300,0),100.0f, -0.003f);
light1->addAnimator(anim);
anim->drop();
```

Creiamo l'*EventReceiver* che gestirà il cambiamento di modalità di rendering da "piatto" a *"Parallax Mapping"*, intercettando la pressione del tasto *SPAZIO* sulla tastiera. Il codice della classe relativa a questo oggetto è riportato di seguito.

```
MyEventReceiver receiver(room, driver);
device->setEventReceiver(&receiver);
```

IRRLICHT DAL DOWNLOAD AL "PLAY-AROUND"

Quattro semplici passi per iniziare. Ecco come eseguire gli esempi utilizzando Visual C++

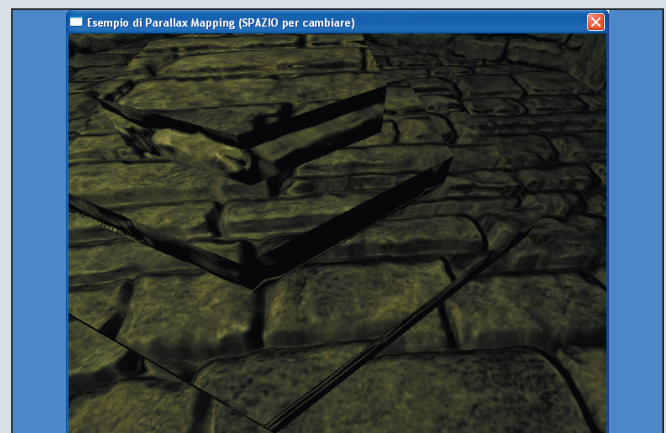
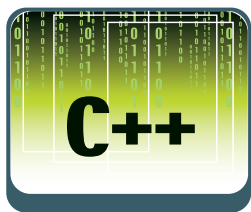
> GLI STRUMENTI

1 La prima cosa da fare per utilizzare IrrLicht è scaricare l'ultima versione del codice dal sito ufficiale *http://irrlicht.sourceforge.net/*, sezione *"Download"*. Estrarre poi il pacchetto in una cartella a piacere.

> I FILE NECESSARI

2 Dal menu *"Tools"* di MSVC++, selezionare *"Options"* e quindi *"Projects"* e *"Directories"*. Da qui inserire le cartelle *"include"* e *"lib"* della cartella principale di IrrLicht sotto le rispettive voci.

> COMPILARE IL CODICE

3 Ora si può tranquillamente copiare il codice trovato sul CD e compilarlo. Tutto dovrebbe andare alla perfezione. Per eseguire con successo il programma prodotto è però necessario copiare nella stessa cartella il file *IrrLicht.dll*

> IL RISULTATO

4 Ci si può muovere all'interno della stanza di mattoni coi tasti freccia e cambiare modalità di rendering con lo SPAZIO. Si noti come le giunture tra i poligoni siano perfettamente piatte, segno che il PM è in ogni caso di una tecnica di tipo 2D.

Impostiamo il titolo della finestra, che ci ricorda che con *SPAZIO* è possibile cambiare la modalità di rendering da piatto a PM e viceversa.

```
// Titolo della finestra
core::stringw str = WINDOW_TITLE;
device->setWindowCaption(str.c_str());
```

Ed infine, il familiare ciclo principale di un programma IrrLicht, con lo *SceneManager* che si occupa di disegnare il tutto a schermo:

```
// Ciclo principale del programma
while(device->run()) {
    if (device->isWindowActive())
    {
        driver->beginScene(true, true, 0);

        smgr->drawAll();

        driver->endScene();
    }
}
device->drop();
return 0;
}
```

Questo completa, in poche righe di codice, un'operazione piuttosto delicata, come quella di implementare un algoritmo di *Parallax Mapping*.
Il codice dell'*EventReceiver* che si occupa di gestire l'input da parte dell'utente è abbastanza semplice ed è riportato di seguito:

```
class MyEventReceiver : public IEventReceiver
{
public:

    MyEventReceiver(scene::ISceneNode* room,
        video::IVideoDriver* driver)
    {
        Room = room;
        Driver = driver;
        actual_material =
                    video::EMT_PARALLAX_MAP_SOLID;

        // Imposta il materiale selezionato dall'utente
        Room->setMaterialType(actual_material);
    }

    bool OnEvent(SEvent event)
    {
        if (event.EventType ==
                    irr::EET_KEY_INPUT_EVENT &&
            !event.KeyInput.PressedDown &&
            Room &&
            event.KeyInput.Key == irr::KEY_SPACE)
        {
            // Cambio il tipo di materiale utilizzato
            actual_material = (actual_material ==
                                video::EMT_SOLID) ?
                video::EMT_PARALLAX_MAP_SOLID :
                                video::EMT_SOLID;

            // Imposto il nuovo materiale
            Room->setMaterialType(actual_material);
        }

        return false;
    }

private:
    video::E_MATERIAL_TYPE actual_material;
    scene::ISceneNode* Room;
    video::IVideoDriver* Driver;
};
```

La funzione *OnEvent()* è richiamata automaticamente da IrrLicht ogniqualvolta si verifica un evento, come ad esempio la pressione di un tasto. Le informazioni relative all'evento (ad es. quale tasto è stato premuto) sono racchiuse nell'oggetto *SEvent "event"*.
La funzione *OnEven()*, di cui si fa overloading, non fa altro che controllare che il tasto premuto sia effettivamente lo *SPAZIO*. In questo caso, con l'istruzione

```
actual_material = (actual_material ==
                                video::EMT_SOLID) ?
            video::EMT_PARALLAX_MAP_SOLID :
                                video::EMT_SOLID;
```

si imposta un valore differente per il campo privato *actual_material*, che determina il materiale effettivamente usato per il rendering della mesh.

CONCLUSIONI

In questo articolo abbiamo visto quali siano i principi su cui si basa la tecnica del *Parallax Mapping*. Abbiamo visto come questa tecnica sia in effetti una approssimazione del risultato "ottimale", che tuttavia permette di ottenere risultati di tutto "rilievo" (è proprio il caso di dirlo!).
L'implementazione che IrrLicht offre del PM è estremamente semplice da utilizzare e può essere impostata dinamicamente, lasciando spazio al programmatore per ulteriori ottimizzazioni o personalizzazioni dell'effetto finale visibile a schermo.

Alfredo Marroccelli

Tutta la potenza dello scripting

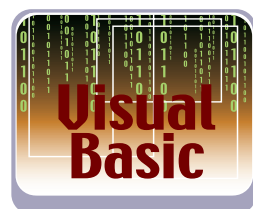
Introduciamo Windows Scripting Host, il linguaggio integrato in Windows che consente di creare potenti script senza bisogno di un compilatore. Vediamo anche come integrarlo in Visual Basic

Windows Script Host è il linguaggio di scripting incluso nei sistemi Windows e che consente di scrivere applicazioni e di eseguirle senza essere dotati di un compilatore. WSH viene utilizzato per realizzare applicazioni che automatizzano operazioni ripetitive senza complicarsi la vita con un compilatore. È dotato tuttavia di oggetti che gli consentono di interagire con Visual Basic oltre che con gli altri software installati nel sistema. Questa alta integrazione con Windows lo rende estremamente comodo in tutte quelle situazioni in cui è necessario eseguire operazioni che altrimenti richiederebbero l'uso di software specifico. Tanto per fare una similitudine si potrebbe immaginare WSH come il vecchio linguaggio *Batch* del Dos. Vediamo come funzione.

UNA DEFINIZIONE RIGOROSA

Windows Script Host - WSH - è il tool che permette di eseguire script dentro i sistemi operativi Windows a 32-bit. WSH è compatibile con *VBScript* e *JScript ActiveX scripting engines* ed ha un proprio modello ad oggetti che può essere referenziato da Visual Basic. WSH permette di eseguire script per Windows Desktop e per il prompt dei comandi ed è l'ideale per la creazione di script non interattivi come quelli di logon, di scheduler e di amministrazione di reti e computer remoti. WSH, essendo basato sulla tecnologia Activex Scripting, può essere visto come un contenitore di oggetti COM anche definiti dall'utente, esso è utilizzabile in tutti i sistemi operativi Microsoft da Windows 98 a Windows XP, in Windows 95, invece, deve essere installato successivamente.
In questo appuntamento, dopo aver introdotto le caratteristiche principali di WSH e della tecnologia ActiveX Scripting sono presentati degli esempi di script che utilizzano gli oggetti WSH dedicati al file system e alla gestione degli oggetti COM. Nella seconda parte dell'articolo, invece, è presentata un'applicazione, basata sul modello ad oggetti WSH per Visual Basic, che amministra degli script. In particolare, essa permette di caricare, eseguire, fare il debug ed archiviare - in un database Access - degli script WSH contenuti in file testo.

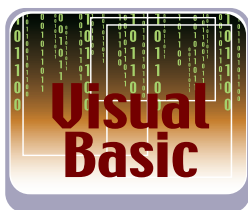
GLI SCRIPT E L'ENGINE WSH

Il motore di WSH consiste di due parti: *WScript.exe* e *CScript.exe*. La prima applicazione esegue script direttamente in Windows la seconda, invece permette di eseguire script dal prompt dei comandi. Di default WSH, è compatibile con VBScript e JScript ActiveX scripting engines, ma tramite applicazioni di terze parti può supportare anche altri linguaggi come PerlScript, PScript e Python. Uno script Windows è un file testo salvato con una delle seguenti estensioni: *js* - per Java Script -, *vbs* - per VB Script, oppure *wsf* - script in formato XML che supporta entrambi i linguaggi. Per implementare uno script, dunque, basta creare un file del Blocco Note, inserire il codice e salvare con l'estensione opportuna. Per eseguirlo in Windows, invece, basta cliccare due volte sul file, questa azione invoca il programma *WScript.exe* che passa a VbScript o JScript Engine lo script, questo lo interpreta e se non ci sono errori lo esegue. Il primo errore trovato viene segnalato, con un *MsgBox.* nel formato: numero di riga e colonna, descrizione errore ecc. Nei nostri esempi utilizzeremo solo file con estensione *.vbs* che permettono di utilizzare la sintassi di VbScript e quindi di Visual Basic. Per eseguire uno script con *CScript.EXE,* dalla finestra *Esegui*

o dal Prompt dei comandi, bisogna scrivere una linea di comando come la seguente:

```
CScript "path/nomescrip.vbs".
```

Infine, segnaliamo che per utilizzare gli oggetti COM in uno script WSH bisogna creare un'istanza dell'oggetto con il metodo *CreateObject.* Questo metodo crea e restituisce un riferimento ad un ActiveX e la sintassi è la seguente:

```
CreateObject(classe,[nomeserver]).
```

Classe è il nome dell'applicazione o dell'oggetto da creare, mentre nomeserver è il nome del Server di rete su cui sarà creato l'oggetto.

NOTE

VERSIONI WSH

La versione più recente di WSH, fornita con Windows XP, è la 5.6. Le altre versioni sono la 1.0, che si trova in Windows 98 e Windows NT, e la 2.0 fornita con Windows 2000. Dal sito http://msdn.microsoft.com/scripting è possibile fare l'upgrade alla versione 5.6. Facciamo notare che non ci siamo persi delle versione ma che WSH è passato direttamente dalla versione 2.0 alla versione 5.6.

WSH OBJECT MODEL

L'*Object Model* di WSH ha 14 oggetti, riassunti nella **Tabella 1**, che consentono di controllare vari aspetti del sistema operativo e delle applicazioni Windows.

Gli oggetti principali sono: *WScript, WshShell* e *WshNetwork*. WScript, come si deduce dalla **Figura 1**, è la radice della gerarchia, esso permette di connettersi e disconnettersi dagli oggetti COM, fermare l'esecuzione degli script, ricavare informazioni sull'ambiente di esecuzione ecc. L'oggetto *WshShell,* invece, ha metodi e proprietà che permettono di creare shortcut, eseguire programmi in locale, manipolare il registro di sistema e le variabili d'ambiente e accedere alle cartelle di sistema.

L'oggetto *WshNetwork,* infine, permette di accedere alle risorse di rete (dischi, stampanti e cartelle condivise), eseguire in rete script locali e ricavare informazioni sugli utenti del sistema.

Oggetto	Utilizzo
Wscript	Permette di accedere ai principali oggetti, metodi e proprietà del *WSH model*
WshArguments	Accede alla collezione degli argomenti forniti dalla linea di comando
WshNamed	Permette di accedere agli argomenti con nome
WshUnnamed	Permette di accedere agli argomenti senza nome
WshNetwork	Utilizzato per accedere alle risorse condivise della rete
WshController	Per creare il processo di uno script eseguito in remoto
WshRemote	Permette di accedere al processo di uno Script remoto
WshRemoteError	Accede alle informazioni sugli errori generati durante l'esecuzione di uno script remoto
WshShell	Permette di accedere alle funzionalità dello Shell di Windows
WshShortcut	Crea *ShortCut* a livello di programmazione
WshSpecialFolders	Permette di accedere alle directory speciali di Windows (*Desktop, MyDocument, Programs …*)
WshURLShortcut	Permette di creare uno *ShortCut* ad una risorsa Internet
WshEnvironment	Permette di accedere alla collezione di variabili di ambiente (*WINDIR, PATH, PROMPT, …*)
WshScriptExec	Permette di stabilire lo stato in uno script che utilizza il metodo *Exec* di *WshShell. Exec* permette di eseguire un'applicazione ed accedere ai canali di input, output ed error

Tabella 1: I principali oggetti di WSH

Fig. 1: *La gerarchia degli oggetti WSH*

DUE SEMPLICI SCRIPT

Presentiamo due script: il primo visualizza la frase *"Hello ioProgrammo",* il secondo interagisce con Microsoft Word creando un documento e aprendo la finestra *SaveAs.* Ricordiamo che il codice di questi script deve essere inserito in dei file testo con estensione vbs.

Il primo script consiste nella seguente istruzione:

```
WScript.Echo("Hello ioProgrammo")
```

Echo è un metodo di WScript che ha le stesse funzionalità del *MsgBox* di VB.

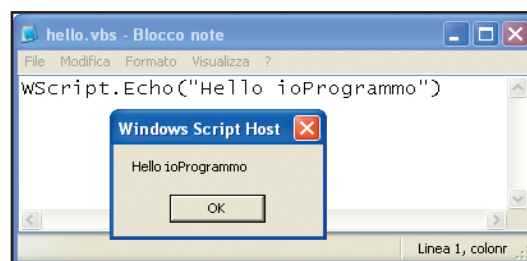



Fig. 2: *Lo script Hello*

Il secondo esempio, più complesso, contiene le seguenti istruzioni.

```
Set Word = CreateObject("Word.Application")
Set Doc = Word.Documents.Add
Word.visible=true
Word.Selection.Text = "Prova file Word"
Const wdDialogFilesaveas=84
Word.Dialogs(wdDialogFilesaveas).Show
```

Le prime istruzioni creano una nuova istanza dell'applicazione Word, aggiungono un docu-

mento all'insieme *Documents* e visualizzano il tutto. Le altre tre istruzioni, invece, inseriscono un testo nel documento ed attivano la finestra di dialogo *"Salva File con nome"*. Si noti che la costante *wdDialogFilesaveas* identifica la finestra *SaveAs* nell'insieme delle finestre di dialogo *(Dialogs)* di Word.

CREARE UNO SHORTCUT A VB6.EXE

Lo script presentato in questo paragrafo permette di creare, sul desktop, uno ShortCut all'applicazione *VB6.exe*. Per lo scopo utilizziamo l'oggetto *WshShell* creato da WScript cioè *WScript.Shell*. Al solito per realizzare lo script è sufficiente copiare il codice in file di testo e salvarlo con estensione *.vbs* Nella parte dichiarativa dello script utilizziamo la *Option Explicit* ed inseriamo delle variabili senza tipo, dato che l'unico tipo supportato da VBScript è il *Variant*.

```
Option Explicit
Private strDesktop
Private wshShell
Private objShellLink
Private PathVB
```

In particolare la variabile *strDesktop* conterrà il path della *special Folder Desktop; wshShell* conterrà un riferimento ad un oggetto *WScript.Shell; objShellLink* è un riferimento ad un oggetto *Shortcut* e la *PathVB* conterrà il path dell'applicazione VB6.
Con le istruzioni seguenti vengono creati gli oggetti *WshShell* e *Shortcut* ed impostati i path del *DeskTop* e dell'applicazione *VB6.exe*.

```
Set wshShell = WScript.CreateObject("WScript.Shell")
strDesktop = wshShell.SpecialFolders("Desktop")
Set objShellLink = wshShell.CreateShortcut(
                                    strDesktop & "\VB.lnk")
PathVB="C:\Programmi\Microsoft Visual
                                    Studio\VB98\VB6.EXE"
```

infine salviamo le caratteristiche principali dello *ShortCut:* destinazione, directory di lavoro, provenienza, tasti di scelta rapida, icona e stile.

```
With objShellLink
 .TargetPath = PathVB
 .WindowStyle = 4
 .HotKey = "CTRL+SHIFT+V"
 .IconLocation = PathVB+", 0"
 .WorkingDirectory = strDesktop
 .Save
End With
Set objShellLink = Nothing
Set wshShell = Nothing
```

Dopo aver eseguito lo script, sul Desktop comparirà l'icona di VB6. Dopodiché la selezione della combinazione di tasti *CTRL + SHIFT+V* o il doppio clic sull'icona avvierà VB6.
Facciamo notare che per creare lo shortcut ad un'altra applicazione, basta cambiare il contenuto della variabile *PathVB*.

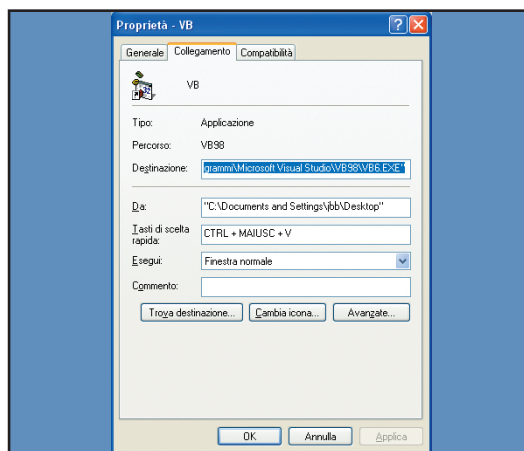
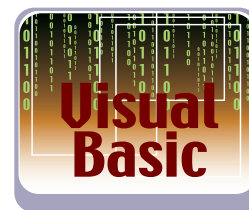



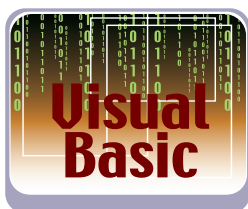
Fig. 3: **La finestra proprietà dello ShortCut**

WSH, SCRIPTCONTROL E VISUAL BASIC

In Visual Basic per utilizzare gli elementi dell'*Object Model* di WSH bisogna referenziare la libreria *"Windows Script Host Object Model."* - *WSHOM.OCX*. In essa, però, non è definito l'oggetto *WScript* e gli oggetti di secondo livello come *WshShell, WshShortcut, WshNetwork, ...,* sono indipendenti e possono essere creati direttamente. Per questo, la maggior parte delle proprietà e dei metodi di WScript non possono essere usati, anche se per alcuni di essi è possibile utilizzare degli elementi alternativi. Per esempio invece di usare il metodo *WScript.Echo* si può usare un *MsgBox*, mentre al posto di *WScript .Sleep* - metodo che sospende l'esecuzione dello script - si può usare l'*API Sleep;* ecc. In ogni caso, per non avere sgradite sorprese, conviene conoscere tutte queste eccezioni, soprattutto quando si vuole riutilizzare uno script WSH in un'applicazione VB. Infine, facciamo notare, che se in un progetto non si vuole referenziare la libreria *WSHOM.OCX*, per creare gli oggetti WSH si può usare la funzione *CreateObject*, come abbiamo fatto nell'applicazione WSH-Visual Basic di esempio. Questa applicazione grazie al controllo *ScriptControl*, può, anche, eseguire script caricati dall'esterno. Descriviamo brevemente questo controllo. *ScriptControl* permette di gestire script

WSH 5.6

L'ultima versione di WSH è stata migliorata sia nel modello ad oggetti che nelle logica di programmazione. Tra le novità introdotte segnaliamo:

- **la gestione della sicurezza,**
- **la possibilità di eseguire script locali in remoto,**
- **la possibilità di determinare lo stato dello standard I/O Streams**
- **l'introduzione degli script wsf.**

Questi ultimi, grazie al supporto del formato XML semplificano la scrittura di script professionali. Lo schema XML adottato, infatti, consente di utilizzare più linguaggi nello stesso file script e di includere script esterni.

che supportano *ActiveX*, per utilizzarlo, in VB, bisogna referenziare l'*OCX MSSCRIPT*. Il controllo fornisce un ambiente di testing in cui è possibile interpretare gli script ed intercettarne gli errori. Esso inoltre, tramite delle opportune proprietà -*Timeout, AllowUI, UseSafeSubset,* ecc -, permette di limitare l'esecuzione di uno script in termini di durata e funzionalità. Del controllo negli esempi utilizziamo il metodo ADDCode, per eseguire il codice dello script, il metodo *ScriptControl1_Error* e l'oggetto *ScriptControl1 .Error* per catturare e visualizzare gli errori di sintassi.

GESTIRE SCRIPT IN VB

In questo paragrafo introduciamo un'applicazione Visual Basic che gestisce file testo che possono contenere script WSH, da eseguire tramite il controllo *ScriptControl*, o testo generico, da elaborare con gli strumenti di MS Word. L'applicazione è composta da un solo form sul quale sono disposti: un *TextBox*, un controllo *ScriptControl* e i comandi che permettono di caricare un file testo (pulsante carica testo), di salvarlo in un database Access (pulsante *Salva*) e di eseguirlo, qualora si tratti, di uno script WSH (pulsante *Run*). Per gestire i testi generici, contenuti nel TextBox, invece, sono previsti i comandi, che permettono di contare il numero di parole (pulsante *Conta*), controllare la sintassi (pulsante *Sintassi*) o creare un documento Word e salvarlo (pulsante *Salva*). Prima di spiegare il codice dell'applicazione descriviamo sommariamente gli elementi che utilizziamo. Iniziamo dal database che conterrà gli script, esso è nominato *Script* e ha solo due campi specificati nella **Tabella 2**.

Campo	Tipo
Codice	Contatore
Testo	Memo

Tabella 2: Il database Script

Questo database - salvato sul desktop - è gestito utilizzando i seguenti elementi *ADODB: Connection* che crea la connessione al database, *Connection.Open* che apre la connessione, *Connection.Execute* che esegue le query e *Recordset* che raggruppa i risultati delle query. In particolare eseguiamo due query una di *Insert* - per salvare lo script - e una di *select* - per interrogare il database in base al campo codice. Di WSH, invece, utilizziamo, direttamente, l'oggetto *WshShell .SpecialFolder*, per selezionare il path del Desktop. Mentre, per aprire e leggere un file testo *(TextStream)* della libreria *WSHOM* utilizziamo gli elementi: *FileSystemObject.OpenTextFile, TextStream.AtEndOfStream* e *TextStream.ReadLine*. Di seguito descriviamo come implementare l'applicazione.

NOTE

INPUTBOX

Per rendere uno script interattivo si può utilizzare il comando InputBox. Esso visualizza un messaggio ed attende l'inserimento di un valore dall'utente. La sintassi del comando è la seguente: Valore=InputBox(prompt[, title] [, default] [, xpos] [, ypos] [, helpfile, context]) Prompt è il messaggio da visualizzare, Title è il titolo, default è il valore di default mostrato nel TextBox, Xpos e Ypos sono le coordinate di un punto dell'InputBox.

IL PROGETTO

Create un nuovo progetto e referenziate la librerie *WSHOM* e il controllo *MSScript.OCX*. Sul form inserite i seguenti oggetti: un textbox; 3 *CommandButton* (nominati *Carica testo, Salva* ed *EseguiScript*); un frame nominato *Word* con all'interno altri 3 *CommandButton* (nominati *Conta, Sintassi* e *SalvaWord*); il controllo *ScriptControl1*. Nella parte dichiarativa del form inserite le costanti che verranno utilizzate con gli oggetti di Microsoft Word.

```
Const wdDialogToolsSpellingAndGrammar = 828
Const wdDoNotSaveChanges = 0
Const wdDialogToolsWordCount = 228
Const wdItalian = 1040
Const wdDialogFilesaveas = 84
```

La procedura che permette di caricare un file testo è la *Carica_Click*. In essa prima vengono creati, con la *New,* gli oggetti *WshShell* e *FileSystemObject,* poi utilizzando il metodo *OpenTextFile,* sulla base del file *prova.txt,* è creato l'oggetto *TextStream,* nominato *listFile*. Quest'ultimo grazie ad un ciclo *While,* impostato su *AtEndOfStream* (sino alla fine del file), con la *ReadLine* consente di caricare, il file *prova.txt,* una riga per volta.

```
Private Sub Carica_Click()
 Set WshShell = New WshShell
 Set FileSystemObject = New FileSystemObject
 strdesktop = WshShell.SpecialFolders("Desktop")
 Set listFile = FileSystemObject.OpenTextFile( _
 strdesktop + "\prova.txt")
 Text1 = listFile.ReadLine
 Do While listFile.AtEndOfStream <> True
  Text1 = Text1 + vbCrLf + listFile.ReadLine
 Loop
End Sub
```

Se il file testo contiene uno script, si può eseguire cliccando il pulsante *Run* che invoca la procedura *EseguiScript*. Quest'ultima dopo aver reso compatibile lo script con Visual Basic - sostituendo il metodo *WScript.Echo* con un *MsgBox* - lo avvia. Se, però, nello script ci sono degli errori sono catturati dall'evento *ScriptControl1_Error.*

```
Private Sub EseguiScript_Click()
 Text1 = Replace(Text1, "wscript.echo", "MsgBox")
 'bisognerebbe gestire le altre eccezioni!
```

```
On Error Resume Next
ScriptControl1.AddCode Text1
End Sub

Private Sub ScriptControl1_Error()
MsgBox "riga:" + CStr(ScriptControl1.Error.Line) _
& " " + " colonna: " _
& CStr(ScriptControl1.Error.Column) + " " _
& ScriptControl1.Error.Description, vbCritical, "Errore"
End Sub
```

Con i pulsanti contenuti nel frame *Word* sono invocate le procedure che permettono di contare le parole e di controllare la sintassi del testo contenute in *Text1*.

```
Private Sub Conta_Click()
Set Word = CreateObject("Word.Application")
Set Doc = Word.Documents.Add
Word.Selection.Text = Text1
Word.Dialogs(wdDialogToolsWordCount).Show
Word.Visible = False
Doc.Close wdDoNotSaveChanges
Word.Quit
End Sub

Private Sub Sintassi_Click()
Set Word = CreateObject("Word.Application")
Set Doc = Word.Documents.Add
Word.Selection.Text = Text1
Word.Selection.LanguageID = wdItalian
Word.Dialogs(wdDialogToolsSpellingAndGrammar)
                                        .Show
Word.Visible = False
If Len(Word.Selection.Text) <> 1 Then
 Text1 = Word.Selection.Text
End If
Doc.Close wdDoNotSaveChanges
Word.Quit
End Sub
```

Per salvare il testo nel database o in un documento *Word* utilizziamo le procedure *SalvaWord* e *Salva* associate agli omonimi pulsanti. Facciamo notare che nel progetto non abbiamo referenziato le librerie ADODB e Microsoft Word, dato che i loro elementi sono creati con la *CreateObject.*

```
Private Sub SalvaWord_Click()
Set Word = CreateObject("Word.Application")
Set Doc = Word.Documents.Add
Word.Selection.Text = Text1
Word.Visible = True
Word.Dialogs(wdDialogFilesaveas).Show
End Sub

Private Sub Salva_Click()
```

```
Set mioshell = CreateObject("WScript.Shell")
strdesktop = mioshell.SpecialFolders("Desktop")
Set objConn = CreateObject("ADODB.Connection")
objConn.Open ("Provider = Microsoft.Jet.OLEDB.4.0;" _
& "Data Source = " + strdesktop + "/script.mdb")
Set objRs = CreateObject("ADODB.Recordset")
query = "INSERT INTO script (  testo ) VALUES (
                              '" + Text1 + "')"
Set objRs = objConn.Execute(query)
objConn.Close
Set objConn = Nothing
End Sub
```

Nel paragrafo successivo è presentato uno script che può essere eseguito dall'applicazione WSH-VB.

UNO SCRIPT INTERATTIVO

Concludiamo l'articolo presentando uno script "interattivo" che seleziona un record del database *Script*. Esso accetta un valore in input (il codice dello script), interroga il database e visualizza il risultato con un Echo (o un MsgBox, dipende da chi lo esegue!). Lo script può essere caricato nell'applicazione WSH-VB, dato che è predisposta per la gestione di script esterni che utilizzano il metodo *WScript.echo*.

```
inpval=InputBox ("inserisci codice:")
Set myshell = CreateObject("WScript.Shell")
strdesktop = myshell.SpecialFolders("Desktop")
Set objConn = CreateObject("ADODB.Connection")
objConn.Open ("Provider = Microsoft.Jet.OLEDB.4.0;" _
& "Data Source = " + strdesktop + "/script.mdb")
Set objRs = CreateObject("ADODB.Recordset")
set objRs = objConn.Execute("SELECT * "_
& "from script where codice=" & inpval )
Do while NOT objRs.EOF
 wscript.echo "codice = " & objRs("codice") _
 & chr(13) & "testo = " & objRs("testo")
 objRs.Movenext
Loop
objConn.Close
Set objConn = Nothing
```

CONCLUSIONI

In questo appuntamento abbiamo presentato alcuni elementi di WSH, uno strumento complesso la cui trattazione non può essere completata in un solo articolo. V'invitiamo ad approfondire gli argomenti introdotti e, magari, discuterne sui Forum di ioProgrammo.

Massimo Autiero

FINESTRE DI DIALOGO DI WORD

Alcune funzionalità di Word sono fornite attraverso delle finestre di dialogo indipendenti, rappresentate dagli oggetti *Dialog* della collezione *Dialogos*. In particolare ogni oggetto dell'insieme *Dialogs* rappresenta una finestra di dialogo predefinita. Per selezionare una di queste finestre bisogna utilizzare la sintassi *Dialogs(index)*, dove *index* è una costante *WdWordDialog* . Nei nostri esempi abbiamo usato le seguenti costanti:

wdDialogToolsWordCount, wdDialogToolsSpellingAndGrammar* e *wdDialogFilesaveas.

L'elenco completo delle costanti lo trovate nell'Help di Word.

SMNP lo 007 del computer

In questo articolo parleremo del protocollo che consente di ottenere ogni tipo di informazioni dalle periferiche collegate al pc. Termineremo con un esempio pratico che controlla lo stato della rete

REQUISITI

Conoscenze richieste
Basi di Visual Basic.NET

Software
Visual Studio.NET

Impegno

Tempo di realizzazione

Ipotizziamo per un istante di voler realizzare un sistema che debba essere in grado di monitorare un certo numero di periferiche permettendoci, nel contempo, di ricevere opportuni allarmi nel caso di malfunzionamento o altro. Pensiamo ad un software che debba analizzare il traffico che passa su una scheda di rete, oppure a un software che determini l'occupazione della memoria o il regime d'uso dell'Hard Disk. Probabilmente, l'architettura che immagineremmo, potrebbe essere costituita da:

- un sistema "centrale" in grado di raccogliere informazioni sullo stato di ogni device da monitorare e capace d'interpretare i messaggi che ci vengono inoltrati da essi;
- un linguaggio comune in grado da fare da ponte tra le periferiche e il sistema centrale.

Il punto è: il sistema chiederà le informazioni sullo stato alle periferiche oppure saranno le periferiche a inviare autonomamente le informazioni al server? E se le due cose non fossero in contrapposizione? Affinché il sistema funzioni in maniera bidirezionale, è necessario che ogni device sappia in che modo interpretare le richieste ricevute dalla centrale e, allo stesso modo, sappia come avvertirla relativamente agli eventi che la riguardano. La bidirezionalità è, ovviamente, una caratteristica necessaria in questi sistemi perché se per qualche ragione il device non riuscisse a comunicare con la stazione centrale per assenza di collegamento o per altre ragioni, quest'ultima deve poter tentare in qualche modo di contattare il device a determinate scadenze di tempo.

IL SOFTWARE

Chiaramente né le periferiche né la stazione centrale sono macchine pensanti. Perciò, una volta stabilito che le due componenti debbano comunicare tramite un protocollo comune, sarà necessario implementare un software che gestisca questo genere di comunicazione: questo tipo di software prende il nome di agente. Conseguenza logica di quest'affermazione è la necessità che ogni device (diverso da tipo a tipo) esponga le proprie caratteristiche che devono, peraltro, essere messe a conoscenza di ogni agente e, ancora, del sistema centrale. Occorre quindi anche un database che contenga la descrizione delle caratteristiche esposte da ogni periferica.

DALLA TEORIA ALLA PRATICA

Nella realtà, l'architettura che normalmente è implementata per raggiungere lo scopo prefisso, non è molto dissimile da quanto prospettato. Volendo dare un nome "proprio" ad ogni componente appena menzionato avremmo:

- **SNMP:** protocollo per la comunicazione tra agenti e sistema centrale;
- **NMS:** sistema centrale;
- **Agente:** sistema software residente su ogni device;
- **MIB:** formato per la descrizione delle caratteristiche di ogni device;
- **Trap:** allarmi, ossia particolari pacchetti inviati dagli agenti all'NMS per la notifica di certe situazioni.

L'acronimo SNMP sta per *Simple Network Management Protocol*, come anticipato poc'anzi, costituisce il protocollo standard (definito dall'*Internet Engineering Task Force* - IETF) per il network management su reti IP. Attraverso esso possiamo controllare qualunque tipo di device che lo supporti, compresi ovviamente router, computer, stampanti, ecc. e, come avremo modo di vedere, molto di più. L'architettura di cui il protocollo *SNMP* fa parte, è definita con

il nome d'Internet *Network Management Framework* (NMF) e consente la gestione dei device "gestibili" attraverso l'utilizzo di agenti ossia moduli software che risiedono su ciascuno di essi. Ciascun agente non fa altro che comunicare con almeno una stazione di gestione centralizzata (*Network Management Station* - NMS) attraverso la quale è possibile leggere e/o scrivere informazioni e raccogliere eventuali segnalazioni *(trap)* inviatele da essi. Le informazioni o caratteristiche che possiamo gestire per una particolare apparecchiatura, mediante il protocollo SNMP, sono dette *Management Objects* e sono contenute all'interno di opportuni file denominati MIB secondo la struttura definita nella SMI *(Structure Management Information)*. SMI è costituita da un insieme di strutture comuni e da uno schema identificativo usato per far riferimento alle variabili del MIB (specificato in *RFC1155*).

IL FORMATO MIB

Il *Management Information Base* può essere visto come un'astrazione di un generico database ed è uno dei concetti più importanti legati al mondo SNMP. Basti pensare che un device del quale non abbiamo disponibile il MIB è paragonabile ad un device che non esiste poiché non sapremmo come interpretare le informazioni da richiedere o ricevute da un agent. Un MIB è semplicemente un file di testo, scritto secondo una notazione particolare (definita come ASN 1.0) che permette di descrivere tutte le variabili alle quali si può accedere via SNMP e relative a quel determinato device. Abbiamo accennato che tutti i MIB, di fatto, fanno parte di una struttura gerarchica (molto simile a quella utilizzata dai DNS per i nomi dei domini) costituita da una radice (senza nome) ed all'interno della quale ogni nodo contiene un identificatore univoco, tipi di dati e modalità di accesso per ciascuna variabile in essi contenuta. Al di sotto della radice di quest'albero, sono inseriti i MIB relativi alle diverse organizzazioni standard e quelli che non risultano standard, solitamente posizionati all'interno del ramo experimental. Le modalità di lettura di una determinata variabile sono analoghe a quanto viene fatto per lo spazio dei nomi DNS. Si parte dalla radice (indicata con un punto) e si scorre l'intera gerarchia sino al raggiungimento dell'informazione voluta. In questa maniera, ogni foglia dell'albero può essere identificata da una sequenza di nomi *(Object Name)* o, analogamente, di numeri (*Object Identifier* - OID) separati da un ".". Ad esempio, *.iso.org.dod.internet .mgmt.mib-2.system* e *.1.3.6.1.2.1.1* identificano la stessa cosa. Ovviamente, ciascun identificativo deve essere necessariamente unico e ciò implica che all'interno di un qualunque sottoalbero non possono esistere più oggetti avente il medesimo OID. Se partiamo dalla radice di questa struttura, scorrendola con l'uso di un qualunque tool adatto a questo scopo, ci accorgiamo presto che esiste il ramo *dod* (6) *(US DEPARTEMENT OF DEFENSE)*, dal quale deriva il nodo internet *(OID=1.3.6.1)*. Al di sotto di esso troviamo diversi nodi tra cui:

- **management:** questo sottoalbero contiene le definizioni delle MIB standard approvate dall'IAB *(Internal Activities Board)*. Attualmente esistono due versioni: la *MIB-I (RFC 1156)* e la *MIB-II (RFC 1213)*, entrambe identificate dallo stesso *OID 1.3.6.1.2.1*.
- **experimental:** utilizzato per identificare oggetti che sono in fase sperimentale;
- **private:** utilizzato per identificare gli oggetti creati dai vari vendor come Cisco, IBM ecc. per la gestione delle variabile che sono specifiche di un loro prodotto.

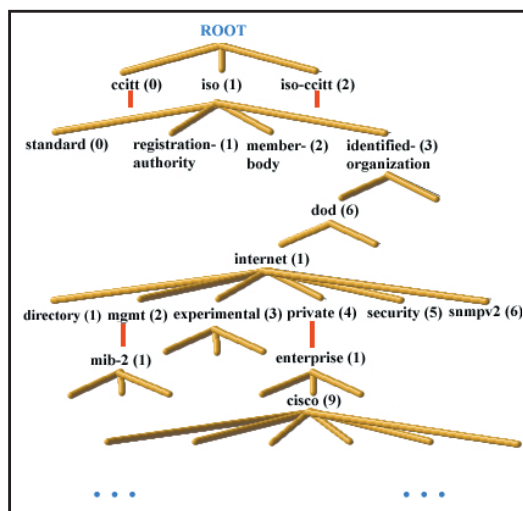



Fig. 1: **La struttura gerarchica che raccoglie (rappresenta) tutti i MIB**

Da quanto sinora detto, appare evidente che un agent deve possedere, per ogni device che intende gestire, il relativo MIB. Tuttavia, è bene sapere che tutti i device disponibili e gestibili attraverso SNMP, devono essere conformi "almeno" alle specifiche definite dal *MIB-II standard (RFC 1213)* prima menzionato ed è a questo che ci riferiremo principalmente in futuro. La comprensione della struttura di un MIB, oltre alla conoscenza del significato delle variabili in esso contenuto, è di notevole importanza per la gestione di un device poiché questo rappresenta la base di conoscenza necessaria al corretto utilizzo delle informazioni da ottenere o inviare agli agent.

COMMUNITY, TRAP, ECC...

L'insieme degli apparati di rete che possono essere

I TUOI APPUNTI

gestiti attraverso l'SNMP appartengono necessariamente ad una community. La comunità rappresenta un identificativo attraverso il quale si cerca di garantire un minimo di sicurezza quando avvengono i colloqui, via SNMP, tra NMS ed agenti. Questo vuol dire che un agent SNMP risponde solamente alla richieste di informazioni effettuate da una NMS appartenente alla sua stessa comunità. Esistono fondamentalmente tre tipi di comunità:

- **monitor:** permette di effettuare solamente interrogazioni agli agent (quindi esclusivamente operazioni di lettura);
- **control:** permette, attraverso gli agent SNMP, di effettuare delle operazioni di lettura/scrittura sul device;
- **trap:** permette ad un agent d'inviare un opportuno messaggio *(trap)* SNMP alla management station per notificarle una determinata situazione.

Bisogna tener presente che i nomi che definiscono una community sono formati da 32 caratteri e sono case sensitive. Esistono dei nomi predefiniti per i diversi tipi di comunità e molto spesso sono public per i casi di sola lettura e private per quelle in lettura/scrittura. Per aumentare il grado di sicurezza, naturalmente, è opportuno modificare queste impostazioni prima possibile. La maniera più valida per imparare a conoscere SNMP, ma soprattutto per vedere se la configurazione delle macchine è a posto, è quella di provare praticamente a leggere alcune informazioni, interrogando direttamente il proprio PC o, meglio ancora, un'altra macchina in rete Per far questo, sono necessari almeno due passi:

- configurazione del protocollo SNMP;
- utility per lo scorrimento delle informazioni;
- utility per l'invio/la lettura delle trap.

Per quanto riguarda il primo passo, faremo riferimento a Windows XP, ma la procedura è analoga anche su altri sistemi operativi. In questo caso, tutto quello che occorre fare è installare il componente SNMP *(Simple Network Management Control)* presente tra i componenti del sistema operativo sotto la

Fig. 2: **Configurazione servizio SNMP**

voce Strumenti di gestione e controllo. Per gli scopi dell'articolo, attiviamo anche la voce *Provider SNMP WMI* che, seppur non necessaria, ci servirà per i discorsi che faremo in seguito. A questo punto assicuriamoci che il servizio SNMP sia avviato con i parametri di default e proseguiamo con il secondo passo. Ora non ci serve che un programma in grado d'interrogare, via SNMP, il PC di destinazione alla ricerca delle variabili dei MIB supportati. Per nostra fortuna non dobbiamo ricorrere a costosi software perché possiamo utilizzare quella denominata *Snmputil.exe,* disponibile all'interno del Windows 2000 */XP Resource Kit* e messa a disposizione anche su Internet (un link dal quale scaricarla è *http://www .petri.co.il/download_free_reskit_tools.htm).* Esiste anche una versione GUI di questo tool denominata *Snmputilg.exe,* ma per gli scopi che ci siamo prefissi sfrutteremo la versione a linea di comando. Ora supponiamo di voler leggere la variabile *sysDescr* (il cui percorso è identificato come *system.1* o, se preferite, *.1.3.6.1.2.1.1.1.1* da questo PC configurato per la community di default *public*). Tutto quello che dovremmo fare è lanciare il comando

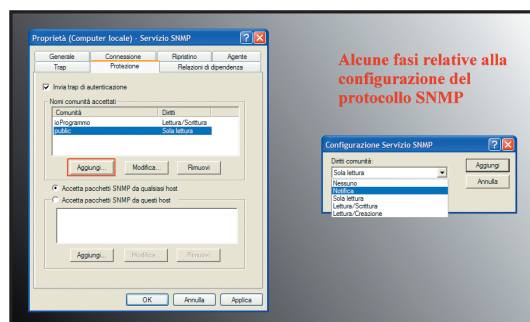
```
Snmputil getnext  127.0.0.1 public system.1
```

Il risultato sarà qualcosa del tipo:

```
Variable=system.sysDescr.0
Value      = String Hardware: x86 Family 6 Model 8
    Stepping 6 AT/AT COMPATIBLE - Software: Windows
         2000 Version 5.0 (Build 2195 Uniprocessor Free)
```

In maniera analoga possiamo leggere altre variabili e cominciare a capire come funziona SNMP. Per facilitare il reperimento delle informazioni, possiamo anche utilizzare alcuni MIB browser in grado di farci navigare all'interno dell'albero gerarchico. Alcuni di essi possono essere "recuperati" dai link suggeriti in fondo all'articolo. A questo punto, abbiamo sicuramente una "infarinatura" sul protocollo SNMP e soprattutto sui principali concetti che ne sono coinvolti. Possiamo passare dunque alla seconda parte di questo cammino.

LA WINDOWS MANAGEMENT INSTRUMENTATION

Credo che il modo migliore per spiegare cosa sia WMI (*Windows Management Instrumentation*) sia quello di citare una frase riportata in un documento ufficiale della Microsoft. Testualmente: *Strumentazione gestione Windows (WMI, Windows Management Instrumentation)* è un componente del sistema operativo Microsoft Windows ed è l'implementazione Microsoft di WBEM *(Web-Based Enterprise*

Management). WBEM è un'iniziativa industriale per lo sviluppo di una tecnologia standard di accesso alle informazioni di gestione in un ambiente aziendale. È possibile utilizzare WMI per automatizzare attività amministrative, quali la modifica del Registro di sistema, in un ambiente aziendale. WMI può essere utilizzato in linguaggi di script che dispongono di un modulo di gestione per Windows e che supportano la gestione di oggetti Microsoft ActiveX.
WMI include un archivio di oggetti conforme agli standard CIM *(CIM Object Manager repository)* ed il servizio *CIM Object Manager* (CIMOM), che gestisce l'acquisizione, la manipolazione di oggetti nell'archivio e la raccolta delle informazioni fornite dai cosiddetti provider WMI, argomento del prossimo paragrafo. Il CIM offre, in definitiva:

- possibilità di gestire gli oggetti come classi astratte. In questo modo, possiamo operare su una qualunque periferica o su di un processo qualsiasi attraverso esse;
- una serie di proprietà e metodi che consentono la gestione delle suddette classi;
- namespace per gestire le classi di oggetti.

In particolare, lo scopo di un namespace è quello di raggruppare tutte le classi e le istanze che riguardano un determinato "sistema" da monitorare. Potremmo anche definirlo come una porzione del CIM *Schema*. In particolare, il *CIMV2 namespace*, che costituisce quello di default per WMI, raggruppa tutte le classi e le istanze che riguardano il sistema Windows locale.

PROVIDER E CONSUMER WMI

Il provider WMI costituiscono un concetto molto importante ed il loro compito è quello di fungere da intermediari tra i componenti del sistema operativo e le applicazioni. Un esempio molto "calzante" per gli scopi di questo articolo, è proprio quello del provider SNMP che fornisce dati ed eventi dalle periferiche SNMP. In sostanza, quando il *CIM Object Manager* riceve una richiesta da un'applicazione *(management application)* per dati che non sono disponibili nel *CIM Object Manager repository* oppure per notifiche di eventi non supportati dal *CIM Object Manager* stesso, inoltra tali richieste direttamente al provider WMI interessato. I provider WMI altro non sono che delle DLL localizzate all'interno della cartella *%SystemRoot%\system32\wbem*. Se date un'occhiata all'interno di questa directory, noterete anche la presenza di file con estensione *.mof*. Un *MOF file* non è altro che un file contenente le definizioni necessarie a registrare un provider WMI (e, quindi, il set di classi che gli appartengono), all'interno del *CIM repository*. Accanto al termine provider, WMI introduce anche quello di consumer. Un consumer altro non è che un'applicazione che sfrutta i dati offerti da un provider WMI e rappresenta quella che in precedenza avevamo chiamato *management application*. In particolare, esso interagisce con le WMI COM API, un set di oggetti COM utilizzabili da diversi linguaggi di programmazione, ivi incluso Visual Basic. I provider WMI vengono installati assieme al sistema operativo, ma la lista di quelli messi a disposizione può essere "allungata" attraverso la definizione di propri. In particolare, abbiamo accennato all'*SNMP Provider* che funge da ponte tra i sistemi di monitoring ed i device che devono essere controllati, consentendo la lettura e la scrittura di MIB SNMP e la rimappatura, in eventi WMI, di trap SNMP. Ovviamente affinché ciò sia possibile, è anche vero che ci debba essere un gruppo di classi, all'interno di WMI, che consenta di rappresentare questo tipo d'informazioni. Per ottenere questo risultato e, quindi, interagire con una qualunque periferica della quale si possieda il proprio MIB, è necessario "inserire" tale database nello schema di WMI affinché le sue caratteristiche siano visibili come oggetti astratti WMI. Benché non ci occuperemo di questo aspetto nell'implementazione del progetto VB, può essere utile sapere che esiste un'utility che ci consente di effettuare questa operazione ed è l'SNMP *Information Module Compiler (smi2smir)* che, opportunamente utilizzata, permette di mappare i MIB voluti nel formato equivalente CIM, consentendoci d'interagire con un determinato device, sfruttando la sua *Management Information Base*, ma attraverso WMI. In particolare, tali informazioni vengono destinate all'interno dell'*SNMP Module Information Repository* (SMIR). L'utilizzo di *smi2smir* è piuttosto semplice:

```
smi2smir /a <Nome file MIB>
```

Attraverso questo semplice comando, non facciamo altro che inserire i dati di un MIB all'interno dell'SMIR per poterli utilizzare all'occorrenza. La cosa interessante da sottolineare, a questo punto, è anche l'esistenza di consumer built-in che consentono di perfezionare il funzionamento di questa architettura. Infatti, WMI dispone di diverse management application in grado di compiere determinate azioni a seguito del verificarsi di alcuni eventi.
In particolare alcuni sono:

- **SMTP Event Consumer:** invia un'email al verificarsi di un determinato evento;
- **Command Line Event Consumer:** avvia un processo al verificarsi di un determinato evento;
- **Log File Event Consumer:** scrive una stringa all'interno di un file di log al verificarsi di un determinato evento.

Tool/Resource Kit
http://www.softpanorama.org/Unixification/Reskit/index.shtml
http://www.microsoft.com/technet/itsolutions/reskits/rktmain.mspx
http://www.netdiscover.org/MibDoc/SN/index.html

SNMP
http://www.ietf.org
http://www.snmplink.org
http://www.freesoft.org/CIE/Topics/108.htm
http://www.rfc-editor.org
http://silver.he.net/%7Errg/NEW-SNMPWORLD/SNMPWORLD.htm
http://www.mibdepot.com/index.shtml
http://www.dataductus.se/snmp/snmplinks.html
http://www.javvin.com/protocol

QUERY, EVENTI E WMI

A questo punto, dobbiamo ancora soffermarci su alcuni aspetti che riguardano WMI per tentare di capire assieme alcuni termini e modi di operare di questo "ambiente". Cominciamo subito con il dire che per poter accedere a oggetti contenuti all'interno del *CIM Repository*, sono necessarie almeno tre informazioni:

- Il namespace ed il nome della macchina ai quali ci riferiremo (ad esempio *root\CIMV2*)
- La classe alla quale ci riferiremo (ad esempio *Win32_Service*)
- L'istanza alla quale ci riferiremo (ad esempio *Win32_Service.Name="SNMP"*)

Il primo passo può essere portato a termine attraverso due possibili modi. Quello che utilizzeremo non fa altro che creare un oggetto *SWBemLocator* ed utilizzando il metodo *ConnectServer* per "indicare" il namespace che ci interessa. Ad esempio:

```
Dim objWmiLocator As New
                    WbemScripting.SWbemLocator
Dim objWmiServices As SWbemServices
'Connessione al namespace root\CIMV2 della
                                macchina locale
 Set objWmiServices=
    objWmiLocator.ConnectServer(".", "root\CIMV2")
```

Il metodo *ConnectServer*, se applicato con successo, ritorna un oggetto di tipo *SwbemServices*. A questo punto, possiamo dire di aver "agganciato" il namespace che ci interessa. Ora dobbiamo interrogare il *CIM Repository* alla ricerca delle informazioni che ci interessano. Per far questo, utilizzeremo WQL in maniera analoga a quanto riportato nell'esempio di sotto:

```
Set ServiceSet=objWmiServices.ExecQuery("SELECT
           State, DisplayName From Win32_Service
                          WHERE State='down'")
```

Questa query WQL non fa altro che cercare, all'interno della classe *Win32_Service*, tutti i servizi down, riportando una collection di elementi, denominata *ServiceSet* (e costituita dallo *Stato* e dal *DisplayName* dei servizi trovati) che può essere successivamente utilizzata per i nostri scopi. A questo punto dovrebbe essere chiaro a tutti che, conoscendo namespace e classi da interrogare, possiamo accedere a moltissime informazioni utili. Ad esempio, possiamo cercare informazioni come "tutte le partizioni del nostro disco rigido con meno del 10% di spazio" oppure "tutti i servizi che sono in modalità *Avvio Automatico*", ecc. Tuttavia è bene porre l'attenzione ancora su di un aspetto interessante ed importante, prima di passare al progetto VB ossia la modalità con la quale viene interrogato il *CIM Repository*. In realtà, WMI consente di eseguire tre tipi di query: sincrone, semisincrone ed asincrone che avremo modo di analizzare nei successivi paragrafi.

IL PROGETTO IN VB

Il nostro programma sarà costituito da una sola form e da alcuni moduli che raccolgono funzioni utili al programma stesso. In particolare nell'articolo riporteremo solo il codice relativo al controllo dello stato dell'interfaccia di rete, ma nel CD allegato il progetto completo contiene alcuni esempi molto interessanti:

- **DISKS:** raccoglie le procedure e le funzioni utili al controllo dello spazio disco libero su tutte le partizioni del proprio disco;
- **PING:** implementa una semplice funzione che consente di "pingare" un host remoto;
- **PORTE:** implementa una procedura che, sfruttando l'*SNMP Provider*, consente il controllo delle porte TCP/IP ed UDP aperte.

All'avvio, l'evento *Load* della form principale compie una serie d'operazioni. Le più importanti riguardano il controllo sulla prima esecuzione del programma e l'impostazione delle variabili che rispecchiano le scelte precedentemente fatte dall'utente. Se il programma è stato avviato per la prima volta, vengono richiamate una serie di procedure che si preoccupano di creare, all'interno del *Registry*, una struttura che memorizza diverse impostazioni utili al corretto funzionamento dello stesso. La chiave che fa da root a queste informazioni è denominata *MasterAgent* (sotto *HKEY_CURRENT_USER\Software\VB and VBA Program Settings*) ed al suo interno troviamo le seguenti sottochiavi:

- **Drives:** per il controllo dello spazio disco sulle partizioni;
- **Eventi:** per il controllo di nuovi eventi nell'*Event Viewer;*
- **Install:** per verificare se si tratta di primo avvio del programma;
- **Interfacce:** per il controllo di interfacce di rete;
- **Processi:** per il controllo sulla creazione o eliminazione di processi;
- **Servizi:** per il controllo di servizi che vanno giù.

Fatta eccezione per la sottochiave *Drives* ed *Install*, le altre controllano semplicemente se l'utente ha o meno attivato quel determinato tipo di controllo e lo attivano all'avvio. La chiave *Drives*, invece, conserva anche altre informazioni tra le quali la soglia di allarme sullo spazio disco rimasto libero,

WMI
http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=6430F853-1120-48DB-8CC5-F2ABDC3ED314
http://msdn.microsoft.com/library/en-us/wmisdk/wmi/wmi_start_page.asp
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmisdk/wmi/wmi_utilities.asp
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmisdk/wmi/sql_for_wmi.asp

espresso come valore percentuale. Una volta determinate le impostazioni lette all'interno del Registry, attraverso un'apposita procedura denominata *ReadFromREG()*, l'evento *Load* della form principale inizia ad impostare le variabili ed i controlli all'interno della form. Successivamente, valorizza un'array denominato *WQLQuery* (che non fa altro che memorizzare le varie stringhe WQL che ci serviranno per i controlli), imposta i flag relativi all'attivazione delle procedure (lanciandole se necessario) e visualizza, finalmente, la form.

IL CONTROLLO DELLE INTERFACCE

A titolo di esempio riporteremo il codice per controllare lo stato dell'interfaccia di rete. Come avremo modo di appurare, le procedure implementate sfruttano l'*SNMP Provider* per ottenere le informazioni che ci servono. Vediamo subito la procedura principale ossia la *StatusInterface():*

```
Public Sub StatusInterface()
  Dim objWmiLocator As New WbemScripting
                                    .SWbemLocator
  Dim objWmiServices As SWbemServices
  Dim objWmiNamedValueSet As New
                 WbemScripting.SWbemNamedValueSet
  'Connessione al namespace della macchina locale
    Set InterfaceSink=New SWbemSink
    Set objWmiServices=objWmiLocator
           .ConnectServer(".","root\snmp\localhost")
    objWmiServices.Security_.ImpersonationLevel=3
    objWmiServices.Security_.Privileges.AddAsString
                              ("SeSecurityPrivilege")
    Set objWmiNamedValueSet=CreateObject(
            "WbemScripting.SWbemNamedValueSet")
    objWmiNamedValueSet.Add
         "AgentReadCommunityName","ioProgrammo"
  'Controlla eventuali interfacce down
    'WQLQuery="SELECT * FROM
        __InstanceOperationEvent WITHIN 1 WHERE
          TargetInstance ISA 'SNMP_RFC1213_MIB_
       ifTable'AND PreviousInstance.ifOperStatus<>
                   TargetInstance.ifOperStatus AND
              TargetInstance.ifOperStatus='down'"
    objWmiServices.ExecNotificationQueryAsync
                     InterfaceSink,WQLQuery(4),,,
                              objWmiNamedValueSet
End Sub
```

Innanzitutto, la prima notazione risiede nel namespace specificato, ossia *\root\snmp\localhost.* A quanto sinora detto, è ovvio che, se avessimo caricato all'interno della struttura esposta da WMI il MIB di un qualunque altro device, avremmo potuto specificare, al posto di localhost, il device considerato. A questo punto va sottolineato il seguente spezzone di codice

```
Set objWmiNamedValueSet=CreateObject(
          "WbemScripting.SWbemNamedValueSet")
objWmiNamedValueSet.Add
     "AgentReadCommunityName", "ioProgrammo"
```

Gli oggetti di tipo *SwbemNamedValueSet* sono molto importanti perché rappresentano delle collection di oggetti *SWbemNamedValue* attraverso i quali è possibile inviare ai provider che si stanno sfruttando, informazioni aggiuntive. In particolare, dunque, nel caso nostro, poiché stiamo sfruttando l'SNMP Provider e considerato che SNMP protegge i pacchetti inviati e ricevuti attraverso il nome della community di riferimento, è assolutamente indispensabile specificare quantomeno questa informazione, impostandolo come ultimo parametro all'interno della chiamata *ExecNotificationQueryAsync*. Ovviamente, nel caso specifico, abbiamo considerato come nome di community (configurata all'interno del servizio SNMP della macchina locale) la stringa ioProgrammo. Ricordate anche che senza questa informazione, il provider SNMP non è in grado di ritornare alcuna informazione e che la query WQL fallirà senza ritornare alcun dato.

CONCLUSIONI

Finalmente siamo arrivati alla fine di questo lungo cammino che speriamo sia risultato interessante alla maggior parte di voi. Ancora una volta, però, desideriamo porre l'accento su di un aspetto dell'articolo piuttosto ovvio: la complessità. SNMP è un argomento complesso quanto potente, se avrete la pazienza di sperimentare, sicuramente otterrete software in grado di controllare praticamente il comportamento di ogni periferica. Prima di lasciarvi, vi ricordiamo che, allegate all'articolo, ci sono altre procedure, non utilizzate nel progetto, ma che potrebbero essere molto utili.

Francesco Lippo

INSTALLAZIONE DELL'WMI SNMP PROVIDER SU WINDOWS XP E WINDOWS 2000

Per i restanti sistemi operativi, consultare la documentazione Microsoft.

Windows XP:

1. Da Pannello di controllo selezionare *Aggiungi/Rimuovi Programmi*
• Selezionare *Aggiungi /Rimuovi Componenti Windows*
• Selezionare la voce *Strumenti di gestione e controllo*
• Selezionare la voce *WMI SNMP Provider* e confermare

Windows 2000:

Avviare il programma *Wbemsnmp.exe* contenuto all'interno della directory *System32 \wbem.*
Se non risulta presente, ricercarlo e decomprimerlo all'interno della cartella *\i386* del CD di Windows 2000.

Software che accetta i Plugin

Presentiamo un metodo che consente di estendere le applicazioni dotandole della possibilità di caricare componenti esterni. Un metodo diffuso quanto utile di fare crescere le applicazioni

L'idea è molto semplice: dotare una nostra applicazione di Plugin. Per *plugin* si intendono una serie di dll che una volta caricate nell'applicazione inseriscono una nuova voce nel menu del programma ed implementano nuove funzionalità. Si tratta di una tecnica comune ormai nella maggior parte dei software e piuttosto utile perché consente di espandere le funzionalità base del programma anche a chi non ne è lo sviluppatore iniziale, posto che chi sviluppa il plugin deve condividere con l'applicazione base una serie di regole che consentono di fare dialogare i due elementi. In questo articolo vedremo come realizzare tutto questo utilizzando gli strumenti messi a disposizione dal .NET Framework: un po' di reflection, un pizzico di design pattern, XML quanto basta e una manciata di codice nel linguaggio che preferiamo. Inizieremo a dare la definizione di cosa è un plugin per arrivare a proporre una possibile architettura e una implementazione in un esempio reale sviluppato in C# e VB.NET (riportata in **Figura 1**).

COS'È UN PLUGIN

Un plugin (o *Add-In*) è un componente software non necessariamente sviluppato dallo stesso team di sviluppo che ha creato l'applicazione che lo ospita *(Host)* e non necessariamente scritto nello stesso linguaggio. Il plugin viene caricato dall'Host a runtime e utilizzato senza che l'applicazione padre debba essere ricompilata. L'obiettivo dell'*addin* è di aggiungere funzionalità all'Host.
Riassumendo, un *plugin*:

1) Può essere scritto da un diverso team di sviluppo.
2) Può essere scritto in un linguaggio diverso.
3) È caricato a runtime tramite late-binding.

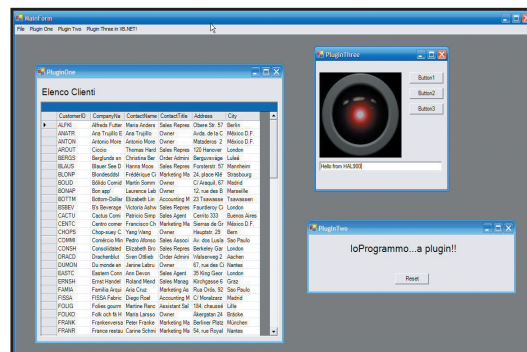



Fig. 1: Screenshot dell'applicazione d'esempio

Il fatto che un plugin possa essere scritto da un team diverso da quello che ha scritto l'applicazione host, significa che quest'ultima dovrà mettere a disposizione una serie di API che altri programmatori possono utilizzare per sviluppare i plugin. Qui il .NET framework ci viene incontro, ed è grazie ad esso che viene garantita la compatibilità binaria tra assembly sviluppati in C# e VB.NET o altri linguaggi che comunque generano codice IL *(Intermediate Language)*. Il fatto che il plugin venga caricato a runtime ci dà la possibilità di "agganciarlo" all'applicazione host senza che questa ne sia a conoscenza a priori (per capirci non esiste un reference diretto all'assembly del plugin durante la compilazione dell'applicazione host). L'idea delle applicazioni che supportano plugin non è una novità e, ad esempio, già da alcune versioni di Microsoft Office è possibile sviluppare *addin* tramite oggetti *COM* che si integrano con le applicazioni della suite. Per essere espandibile, la nostra applicazione dovrà avere qualche meccanismo per ricercare, caricare e attivare i plugin che sono stati sviluppati appositamente per

essa. Prima di addentrarci nei dettagli architetturali e implementativi dei plugin facciamo un breve ripasso sul concetto di *reflection* nel .NET Framework.

LA REFLECTION

La reflection è una tecnica che permette di interrogare i metadati di un assembly a runtime e utilizzare le informazioni recuperate per istanziare oggetti, eseguire metodi ed accedere alle proprietà il tutto a runtime. Ne parliamo approfonditamente in questo stesso numero di ioProgrammo nell'articolo di Antonio Pelleriti. La *reflection* quindi torna utile se volessimo caricare a runtime un *assembly* e mandarlo in esecuzione cosa che ci prefiggiamo di fare in questo articolo. Delle classi del framework che si trovano nel namespace *System.Reflection* quelle di maggior interesse per la trattazione di questo articolo sono:

- *System.Type*
- *System.Reflection.Assembly*
- *System.Reflection.MethodInfo*
- *System.Reflection.PropertyInfo*

La classe *Assembly* rappresenta il punto di partenza, essa contiene i metodi per recuperare informazioni su un assembly e su come caricarlo all'interno dell'*AppDomain* della nostra applicazione. Una volta ottenuto un reference all'assembly possiamo utilizzare le altre classi per istanziare oggetti eseguire metodi o semplicemente recuperare informazioni sui metodi o sulle proprietà. Tutti gli oggetti del CLR dispongono del metodo *GetType* (ereditato da *Object*) che ritorna il tipo che rappresenta l'oggetto (un'istanza di *System .Type*). Tramite l'oggetto *Type* è possibile recuperare altre informazioni quali i metodi implementati (sia pubblici sia privati) le proprietà esposte, i campi, gli attributi, ecc…
Ad esempio il metodo *GetMethods()* della classe *Type* ritorna un array di oggetti *MethodInfo* che contiene informazioni su un metodo della classe e che permette tramite il suo metodo *Invoke* di chiamarlo passando gli eventuali parametri. Per capire l'utilità della reflection analizziamo il seguente esempio che mette in luce le funzionalità principali:

```
Assembly asm = Assembly.LoadFrom(fileName);
Object obj = asm.CreateInstance("Classe1", false);
Type type = obj.GetType();
type.InvokeMember("FaiQualcosa",
              BindingFlags.InvokeMethod, null, obj, null);
```

Tramite queste righe di codice viene caricato un assembly (*DLL* o *EXE*) dal file system (il nome è memorizzato nella variabile *fileName*), si istanzia un oggetto di tipo *Classe1* (contenuto nell'assembly appena caricato) e viene eseguito il metodo *FaiQualcosa*. Queste poche righe di codice sono del tutto equivalenti a queste (avendo opportunamente referenziato l'assembly in cui è definita *Classe1*):

```
Classe1 c1 = new Classe1();
c1.FaiQualcosa();
```

La differenza sta nel fatto che il nome del metodo e il nome della classe sono contenuti in una stringa e quindi abbiamo la possibilità di definirli a runtime magari leggendoli da un file di configurazione. Questo è quello che in gergo si chiama *late-binding*, cioè il nome delle classi e dei metodi da eseguire sono definiti solo a runtime. Utilizzeremo qualcosa di molto simile per istanziare i plugin caricati dell'applicazione di esempio che vedremo tra poco.

LE TRE FASI

Uno dei problemi che la reflection risolve è la possibilità di implementare il *"late binding"* cioè il caricamento e l'esecuzione a runtime di un tipo (a differenza dell'*early binding* in cui il tipo e i metodi da chiamare sono definiti staticamente nel nostro codice). Come dicevamo durante l'introduzione per far sì che la nostra applicazione carichi ed esegua correttamente i plugin previsti è ncessario che svolga tre compiti ben distinti:

1) *Ricerca dei plugin*
2) *Caricamento*
3) *Attivazione e utilizzo dei plugin.*

RICERCA DEI PLUGIN

La prima operazione che l'applicazione host deve compiere è capire dove sono e quali sono i plugin da caricare. Ci sono vari modi per gestire questa informazione. La più semplice consiste nel definire una cartella del file system in cui andare a pubblicare tutti i plugin e far sì che l'applicazione host effettui una ricerca e provi a caricare tutti gli assembly che trova. In questo modo però si corre il rischio di caricare DLL che non sono plugin. Infatti, se per sbaglio nella cartella viene copiato un file che non è un plugin, l'applicazione host

se ne accorgerà dopo il caricamento generando un'eccezione. Il grosso problema è che tutti gli assembly che non sono plugin rimangono in memoria inutilizzati senza che ci sia modo di scaricarli. Per risolvere questi problemi possiamo ricorrere ad un file di configurazione (un file XML) che contiene la lista dei file da caricare. La nostra applicazione legge il file XML e ottiene la lista dei plugin. In questo modo si risolve il problema del caricamento non valido e ci permette anche di pubblicare i plugin nella cartella che preferiamo. Nell'applicazione di esempio allegata al presente articolo si è optato per l'utilizzo di un file xml contenente le informazioni dei plugin da caricare, utilizzando il sequente schema:

```
<?xml version="1.0" encoding="utf-8" ?>
<Plugins>
  <Plugin id="1">
    <FriendlyName>PluginOne</FriendlyName>
      <ClassName>ioProgrammo.PluginOne
                                    </ClassName>
    <Dll>D:\pluginone.dll</Dll>
  </Plugin>
</Plugins>
```

Le informazioni che decidiamo di inserire nel file XML possono essere molteplici.
In questo esempio si è deciso di memorizzare un *"FriendlyName"* per il plugin da caricare (una sorta di nome che lo identifica), il nome della classe da istanziare e il nome completo di path della DLL. La scelta di includere anche il nome della classe seppur non indispensabile ci facilita il compito di ricerca (in caso contrario sarebbe necessario effettuare una ricerca su tutte le classi contenute nella DLL per trovare quella che rappresenta il plugin).

NOTA

La *reflection* permette di ottenere informazioni sugli oggetti e i suoi metodi a runtime utilizzando il nome dell'oggetto e il nome del metodo. Maggiori dettagli li potete trovare partendo da qui:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystemreflection.asp

CREARE UN PLUGIN

**L'esempio allegato implementa una semplice applicazione che supporta i plugin sfruttando i meccanismi presentati nell'articolo.
Oltre all'applicazione host vengono implementati alcuni plugin sia in linguaggio C# che in VB.NET proprio per mostrare l'indipendenza dal linguaggio di programmazione. Per l'esecuzione dell'esempio è necessario aprire il file della soluzione con VS.NET 2003 compilare l'applicazione ed mandarla in esecuzione. Dalla finestra principale scegliere il menu *File | Load Plugin.***

CARICAMENTO

Una volta individuato l'assembly o gli assembly da utilizzare tramite il file XML, dobbiamo caricarli all'interno dell'*AppDomain* dell'applicazione host. Il metodo utilizzato per il caricamento è quello illustrato nell'esempio precedente, a disposizione abbiamo il nome del file completo di percorso letto dal file di configurazione XML, possiamo dunque usare il metodo *Assembly.LoadFrom(nomeFile)* che ritorna l'istanza dell'assembly caricato.

ATTIVAZIONE

La terza fase prevede l'attivazione dell'assembly. L'attivazione può essere a sua volta suddivisa in due fasi: *Creazione di un'istanza della classe* ed *Esecuzione di un metodo di inizializzazione* (le due fasi possono coincidere se il metodo di inizializzazione è il costruttore della classe). La creazione dell'istanza della classe si effettua utilizzando il metodo *Assembly .CreateInstance(nomeDellaClasse)* che ritorna un'instanza dell'oggetto. Una volta ottenuta l'istanza dell'oggetto non resta che chiamare qualche metodo...si ma che metodo? Come può l'applicazione host conoscere il nome del metodo da chiamare per "attivare" il plugin? La soluzione ci viene dal paradigma *OO*. Tra i concetti della programmazione *Object Oriented* troviamo l'ereditarietà e le interfacce.
L'ereditarietà prevede che un oggetto derivi da un altro oggetto estendendone o specializzandone le funzionalità, le interfacce invece sono un meccanismo per garantire che una classe abbia una serie di proprietà e metodi imposti dall'interfaccia stessa. La differenza tra ereditare da una classe base e implementare un'interfaccia sta nel fatto che l'ereditarietà rappresenta una relazione *IS-A* (la classe che eredità è una *classe Base*) mentre l'implementazione di una interfaccia rappresenta una relazione *CAN-DO* (la classe che implementa le interfacce deve fare quello che l'interfaccia impone). Un'altra differenza tra le due è che quando si eredita da una classe si possono sfruttare le funzionalità già implementate nella classe base mentre un'interfaccia non implementa alcun metodo ne dichiara solo la firma (nome, valore di ritorno e parametri). L'utilizzo delle interfacce nel caso di plugin è da preferire all'ereditarietà proprio perché il plugin deve rispettare il contratto stabilito dall'applicazione host, solo in questo caso l'applicazione host può utilizzare il plugin. Inoltre l'uso delle interfacce rende più flessibili e indipendenti i plugin che non si trovano a dover supportare meccanismi ereditati dalle classi predisposte dall'applicazione host. Ora che abbiamo gli strumenti necessari alla realizzazione di una applicazione che supporta i plugin presentiamo una possible soluzione.

L'ARCHITETTURA DELL'APPLICAZIONE

In **Figura 2** è mostrata l'architettura dell'applicazione d'esempio allegata all'articolo.
L'applicazione host è composta di due classi fondamentali, una *MainForm* che rappresenta la finestra che conterrà i plugin e una classe addetta alla gestione dei plugin.

Fig. 2: ***Architettura dell'applicazione***

Il meccanismo di caricamento è implementato nella classe *PluginManager* che utilizza un file XML per l'identificazione dei plugin da caricare. I plugin *(PluginOne, PluginTwo)* come detto nel paragrafo precedente implementeranno l'interfaccia *IpluginInterface* così definita:

```
public interface IPluginInterface
{
        String Name {get;}
        void Initialize(Form parentForm);
        MenuItem GetMenu();
}
```

L'interfaccia *IPluginInterface* "obbliga" i plugin ad implementare i metodi *Initialize* e *GetMenu* oltre alla proprietà *Name*. Il metodo *Initialize* serve al plugin per eseguire il codice di startup necessario (inizializzazione di variabili, lettura dei settaggi, ecc…). Inoltre siccome tutti i plugin implementano l'interfaccia *IPluginInterface* l'oggetto *PluginManager* può sfruttare questa conoscenza per invocare i metodi che sicuramente saranno implementati. La logica dell'applicazione dovrebbe oramai risultare abbastanza chiara. L'applicazione host quando richiesto (o durante il caricamento se preferiamo farlo subito) istanzia un'oggetto di tipo *PluginManager* e si fa restituire l'elenco dei plugin da utlizzare. Una volta ottenuto l'elenco chiamerà per ogni plugin il metodo *Initialize* passando un'istanza di se stesso (istanza che il plugin può utilizzare per ottenere informazioni sull'applicazione che lo ospiterà) e successivamente chiamerà il metodo *GetMenu* per farsi restituire l'elenco dei menu da "agganciare" al suo menu principale (o al sottomenu plugin se esiste).

```
PluginManager pluginManager = new
                        PluginManager(pluginFile);
pluginManager.LoadPluginData();
for(int i=0;i<pluginManager.Count;i++)
{
   IPluginInterface plugin =
                        pluginManager.GetPlugin(i);
   plugin.Initialize(this);
   MenuItem menu = plugin.GetMenu();
   if (menu != null)
      mainMenu.MenuItems.Add(menu);
}
```

A questo punto, terminata la fase di inizializzazione l'applicazione procede ignorando la presenza dei plugin che interverranno quando l'utente seleziona uno dei menu creati dai plugin stessi. Sono infatti loro ad eseguire il codice del gestore degli eventi scatenati dai menu.

L'APPLICAZIONE D'ESEMPIO

Allegato all'articolo si trova un progetto composto da un'applicazione host e un certo numero di plugin sviluppati in C# o VB.NET che sebbene non abbiamo alcuno scopo reale, sono un ottimo esempio a supporto del presente articolo.

CONCLUSIONI

Come avete avuto modo di vedere, la realizzazione di applicazioni a plugin è di per sé abbastanza semplice ma allo stesso tempo efficace al raggiungimento dello scopo.
Le potenzialità dell'architettura presentata risolvono parecchie situazioni in cui è richiesto che la nostra applicazione Windows forms sia estendibile da altri sviluppatori. La soluzione in questo articolo è volutamente semplice per mettere in evidenza i concetti base.
Estendendo l'interfaccia *IPluginInterface* aggiungendo ulteriori metodi e proprietà si possono aumentare le funzionalità e l'integrazione tra l'applicazione host e i plugin. Allo stesso modo se l'applicazione host mette a disposizione una gamma più ampia di oggetti per l'interazione, i plugin possono rendere l'applicazione ancor più personalizzata.

Emanuele Del Bono

La differenza tra ereditare da una classe base e implementare un interfaccia sta nel significato: quando eredito creo una relazione *IS-A* con la classe base quando implemento un'interfaccia creo una relazione *CAN-DO*. Il fatto di implementare un'interfaccia vuol dire che la classe accetta il contratto imposto dall'interfaccia. Inoltre ereditando da una classe base ne eredito anche gli eventuali metodi/proprietà /campi protetti cosa che non avviene nel secondo caso visto che un'interfaccia non implementa codice.

Input/Output ad alte prestazioni

Uno tra i principali fattori che influiscono sulle performance di un'applicazione è sicuramente la gestione del I/O. Vediamo come ottimizzare tali operazioni attraverso l'uso delle API NIO

Le operazioni principali svolte da un calcolatore sono l'Input/Output e l'elaborazione. Spesso il compito principale è costituito dall'I/O, mentre l'elaborazione è semplicemente accessoria. La gestione dell'I/O a livello applicativo rappresenta un punto cruciale per quanto riguarda l'analisi e la progettazione software, in quanto influisce pesantemente sull'usabilità e sulle prestazioni del sistema stesso. Ovviamente, oltre ad un'accurata fase di analisi ed implementazione è fondamentale saper scegliere la tecnologia che meglio si addice alla risoluzione di tali problemi. Fino a qualche anno fa, gli strumenti messi a disposizione da Java per la gestione dei meccanismi di I/O risultavano abbastanza carenti e limitati, pertanto il programmatore che si apprestava a sviluppare parti di codice riguardanti la comunicazione tra dispositivi diversi, si trovava spesso davanti ad un muro invalicabile rappresentato dalle mancanze della tecnologia sottostante. Con il rilascio della JVM 1.4, alias Merlino, molti problemi riguardanti l'I/O sono diminuiti, se non del tutto annullati. Infatti, la migliore "magia" apportata da Merlino ha riguardato proprio l'introduzione di un nuovo sistema per la gestione dell'I/O: NIO. In un primo momento i progettisti Sun avevano battezzato tale novità con il nome *"Java's I/O System Version 2"*. Quest'ultimo è stato poi convertito in NIO, che sta a significare *"New Input /Output"*, in quanto non rappresenta un'evoluzione, bensì una completa rivoluzione del precedente meccanismo. Secondo alcuni programmatori esperti, ed in particolar modo quelli legati alla piattaforma enterprise, NIO, che è la risposta al JSR 51 del Java *Community Process*, rappresenta l'introduzione più importante nel linguaggio Java dall'avvento delle *Java Foundation Classes /Swing*. Nel corso di questo articolo verranno descritte le maggiori features introdotte dalle API NIO, con particolare attenzione alle tematiche inerenti la manipolazione dei file, attraverso una combinazione di concetti teorici ed esempi pratici, in modo da dare al lettore una panoramica quanto più esaustiva di questa nuova libreria. Inoltre, verranno effettuati alcuni confronti con la libreria di I/O originale in modo da stimare più dettagliatamente la bontà delle novità apportate.

NIO: ECCO LE NOVITÀ

Le classi riguardanti le API NIO sono posizionate nei seguenti packages:

- *java.nio.**
- *java.nio.channels.**
- *java.nio.charset.**

Analizzando la documentazione delle classi presenti in questi packages è facile intravedere le prime migliorie apportate dagli ingegneri Sun: una più accurata strutturazione delle classi e l'introduzione di metodi che aiutano il programmatore nella risoluzione di alcuni problemi. Come già annunciato in precedenza, l'introduzione di NIO rappresenta un modo del tutto nuovo per quanto riguarda la gestione dei meccanismi di Input/Output all'interno del mondo Java. Questa totale eterogeneità con la libreria di I/O originale è ancora più intuibile se ci si addentra nel cuore dei due sistemi: lo scambio dati. Si passa, infatti, da una trasmissione dati di tipo stream-oriented ad una block-oriented. La prima metodologia, utilizzata nell'astrazione originale della gestione dell'I/O, è caratterizzata da uno scambio dati che avviene un byte

alla volta. Di contro, lo scambio dati basato sui blocchi, utilizzato nell'implementazione delle API NIO processa le informazioni in blocchi di dati. La soluzione orientata agli stream, sebbene sia molto sofisticata, tende ad essere generalmente, ma non necessariamente, più lenta di quella a blocchi. Questo miglioramento prestazionale presenta però lo svantaggio di rendere il codice meno elegante rispetto a quello tradizionale. Un'altra importante caratteristica di NIO è rappresentata da alcune ottimizzazioni effettuate a basso livello: l'implementazione di alcune attività cruciali, come ad esempio la gestione dei buffer, non è più demandata a codice nativo specifico da una piattaforma all'altra, ma vengono utilizzate delle procedure messe a disposizione dal sistema operativo su cui si esegue la Java Virtual Machine. Anche in questo caso i benefici ottenuti impattano in maniera positiva sulle prestazioni. Queste nuove features appena descritte sono molto utili per il programmatore in quanto aiutano a comprendere meglio il funzionamento interno di NIO, ma non sono assolutamente indispensabili poiché nascoste dall'astrazione e non strettamente collegate agli aspetti di programmazione. Nel prosieguo dell'articolo verranno trattati in maniera più approfondita gli aspetti prettamente implementativi della nuova gestione dell'I/O, in modo da far acquisire al lettore le basi che lo porteranno allo sviluppo di applicazioni ad alte prestazioni.

BUFFERS & CHANNELS

Le basi dell'Input/Output sono improntate sui buffer e su come essi sono gestiti. Nelle API NIO i buffer sono rappresentati dalla classe *java.nio.Buffer,* mentre il loro "smistamento" da un dispositivo all'altro viene demandato alle classi residenti nel package *java.nio.channels.* Sostanzialmente i buffers non sono altro che contenitori di dati sequenzialmente lineari su cui leggere e scrivere, mentre i channels rappresentano un condotto di collegamento tra le varie entità (file, socket, pipe). Per chi ha già lavorato con il sistema di I/O tradizionale, il concetto di channel può essere accostato a quello di stream, con l'enorme differenza che la comunicazione su channel è di tipo bidirezionale mentre quella basata sugli stream è unidirezionale. Se ad esempio volessimo realizzare delle operazioni di lettura e scrittura su un file non ci sarà più il bisogno di lavorare con un oggetto di tipo *FileInputStream* ed un altro di tipo *FileOutputStream:* attraverso *FileChannel* si potranno eseguire entrambe le operazioni ed anche qualcosa in più che scopriremo nel corso dell'articolo. Scendendo ancora di più nel dettaglio possiamo definire un oggetto di tipo *Buffer* come un array di dati di dimensione fissa. Come si può facilmente notare in **Figura 1**, NIO mette a disposizione dello sviluppatore una specializzazione di *Buffer* per ogni tipo primitivo presente nel linguaggio, fatta eccezione per quello booleano.

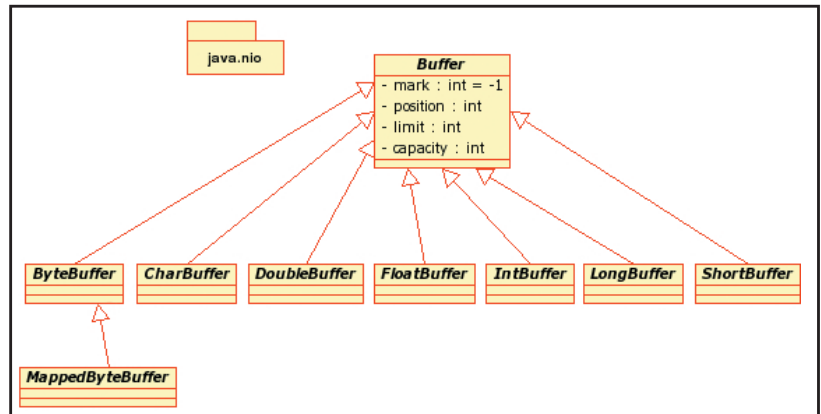



Fig. 1: ***Class diagram relativo ai buffer contenuti nel package java.nio***

Tutti i *Buffer* presentano i seguenti attributi fondamentali:

- **capacity:** rappresenta il numero massimo di elementi che vi possono essere contenuti. È impostato al momento dell'allocazione del buffer stesso e non può essere né modificato né negativo.
- **limit:** è il limite oltre il quale non dovrebbero essere effettuate operazioni di lettura e scrittura. Non può essere né negativo né tantomeno maggiore della capacità.
- **position:** è il puntatore al prossimo elemento da leggere/scrivere. Questo valore viene automaticamente aggiornato alle chiamate dei metodi *get()* e *put()*.
- **mark:** contrassegna una specifica posizione nel buffer. L'invocazione del metodo *reset()* esegue la seguente istruzione: *position = mark.*

Secondo quanto dichiarato sopra, la seguente condizione dovrà sempre essere soddisfatta:

```
0 <= mark <= position <= limit <= capacity
```

Come tutti gli oggetti definiti dal linguaggio Java, prima di iniziare ad usarli è necessaria

OPEN SOURCE E NIO

Per meglio comprendere l'utilizzo di NIO è possibile consultare i sorgenti dei seguenti progetti Open Source che ne fanno uso:

Reattore
http://reattore.sourceforge.net

Kowari
http://www.kowari.org

Raining Sockets
http://jniosocket.sourceforge.net/

UberMQ
http://www.ubermq.com/

Free chatserver
http://freecs.sourceforge.net/

The Bamboo Distributed Hash Table
http://bamboo-dht.org/

una fase di creazione. Ogni specifico *Buffer* presenta i suoi metodi statici di factory per effettuare tale operazione, ma le differenze tra le varie estensioni sono minime. Pertanto, i seguenti esempi di creazione di un *DoubleBuffer* potranno essere utilizzati, con qualche piccolo accorgimento, per la creazione di altri tipi di *Buffer:*

```
// allocazione di un buffer di 5 elementi per dati di
                                              tipo double
DoubleBuffer bufferEmpty = DoubleBuffer.allocate(5);
// creazione di un buffer da un array esistente
double[] doubleArray = { 15.6,  28.33, 0.978, 73.82,
                                                 14.181 };
DoubleBuffer bufferArray =
                          DoubleBuffer.wrap(doubleArray);
// creazione di un buffer con capacity = 5,  position
                                      = 1 e limit = 3
DoubleBuffer bufferArrayOffset =
                       DoubleBuffer.wrap(doubleArray, 1, 3);
```

Queste modalità di allocazione, il cui risultato è mostrato in **Figura 2**, sono di tipo non diretto in quanto non fanno altro che operare su reference di array precedentemente definiti.

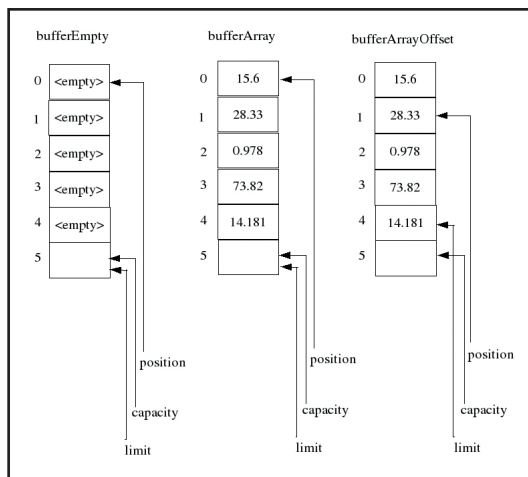



Fig. 2: ***Struttura dei buffer dopo il processo di allocazione***

È lampante che questo tipo di soluzione non apporta alcun incremento di performance. Se si ha bisogno di migliorare le prestazioni viene infatti suggerito l'uso dei buffer di tipo diretto. Tale modalità, dipendente dalla piattaforma su cui si esegue la JVM, è consentita solo con l'uso di oggetti di tipo *ByteBuffer*. I benefici che ne conseguono sono per lo più riguardanti gli aspetti legati alla memoria: i dati vengono memorizzati in una struttura che risiede al di fuori della heap. Pertanto il garbage collector non dovrà preoccuparsi di effettuare operazioni di allocazione o deallocazione su tali oggetti. Di seguito viene mostrato un esempio di allocazione diretta:

```
// allocazione di un buffer diretto di 1024 bytes
ByteBuffer directBuffer =
                         ByteBuffer.allocateDirect(1024);
```

L'uso di questa tecnica è raccomandato soprattutto quando ci si trova a lavorare con buffer di grandi dimensioni in modo da diminuire il carico di memoria. Nella sezione successiva parleremo di come incrementare le prestazioni anche in termini di velocità quando si trattano dati contenuti all'interno di file.

FILE MAPPATI IN MEMORIA

MappedByteBuffer è sicuramente la più importante innovazione, riguardante le operazioni su file, introdotta con NIO. Questo buffer, che estende direttamente *ByteBuffer* (vedi **Figura 1**), stabilisce un collegamento diretto verso l'area di memoria virtuale del sistema dove risiede il file su cui si vogliono effettuare operazioni di lettura e/o scrittura. Sebbene *MappedByteBuffer* rappresenti un'enorme novità nel mondo Java, va evidenziato che questa tecnica non fa altro che utilizzare delle funzionalità oramai presenti all'interno di tutti i sistemi operativi moderni. Infatti, questo nuovo modo di gestire i file si limita ad invocare particolari system call, solitamente identificate dal nome *nmap().*
L'accesso ai file eseguito con questa tecnica, oltre ad apportare i miglioramenti in memoria già visti con l'uso dei buffer diretti, ne incrementa le prestazioni in termini di velocità. È importante, però, sapere che tale approccio comporta anche dei potenziali rischi: non esiste una separazione tra modifica e salvataggio di un dato. Infatti, quando si esegue un'operazione di scrittura con un *MappedByteBuffer* si va direttamente a modificare il file residente sul file system.
Di seguito vengono elencate le istruzioni necessarie a mappare un file in memoria:

```
// apre una connessione di tipo input stream verso
                                        un file specifico ...
FileInputStream fis = new
          FileInputStream("/home/fortino/mioFile.txt");
// ... e ne ottiene il canale associato
FileChannel channel = fis.getChannel();
int size = (int)channel.size();
// mappa l'intero file in memoria in modalità read/write
MappedByteBuffer mb = channel.map(
```

```
FileChannel.MapMode.READ_WRITE, 0, size);
```

Il precedente frammento di codice esegue il mapping in memoria dell'intero file. Le modalità di mapping sono descritte dalla classe *FileChannel.MapMode* e sono così definite:

- **READ_WRITE**: tutti i cambiamenti effettuati sul buffer verrano direttamente propagati al file collegato al canale.
- **READ_ONLY**: permette solo operazioni di lettura *(get())* sul buffer. Eventuali operazioni di scrittura (*put()*) propagheranno eccezioni di tipo *ReadOnlyBufferException*.
- **PRIVATE**: le modifiche non verrano propagate al file, ma solo al buffer. Tale modalità, detta *copy-on-write*, è messa a disposizione dai vari sistemi operativi.

COPIA DI FILE

Dopo una fase di conoscenza principalmente concettuale delle API NIO, passiamo ad un'altra più operativa, prendendo in esame l'esempio più comune: la copia di un file residente sul file system. La prima azione da eseguire consiste nel procurarsi un collegamento verso tale file in modo da ottenerne il contenuto. L'esempio illustrato nel precedente paragrafo è utilizzabile per ottenere tale risultato, ma di seguito verrà descritto un modo alternativo, che ha lo scopo di memorizzare il contenuto del file in un *ByteBuffer* di tipo diretto:

```
// crea due random access file stream ...
RandomAccessFile raf = new RandomAccessFile(
        "/home/fortino/mioFile.txt", "r");
RandomAccessFile rafCopy = new RandomAccessFile(
        "/home/fortino/mioFile_copia.txt", "w");
// ... e ne ottiene i canali associati
FileChannel readChannel = raf.getChannel();
FileChannel writeChannel = rafCopy.getChannel();
// alloca un buffer diretto da 1024 bytes per copiare
        i dati da un canale all'altro
ByteBuffer directBuffer =
        ByteBuffer.allocateDirect(1024);
```

Dopo la creazione degli elementi necessari al raggiungimento del nostro obiettivo, vediamo come trasferire i dati da un canale all'altro:

```
// legge il file iterando sul canale di lettura
while(readChannel.read(directBuffer) > 0) {
    directBuffer.flip( ); // prepara il buffer per la fase
            di scrittura in un altro canale
    writeChannel.write(directBuffer); // scrive il buffer
            sul canale di destinazione
    directBuffer.clear( ); // resetta il buffer
}
// chiude il canale di lettura e quello di scrittura
readChannel.close( );
writeChannel.close( );
```

L'esempio appena visto, oltre ad essere una valida soluzione per la copia del contenuto di un file, ha come scopo quello di far acquisire al lettore maggiore dimestichezza nell'uso di *Buffer* e *Channel*. Infatti, NIO offre un set di metodi di utilità che vanno incontro al programmatore nella risoluzione di problemi comuni, come appunto il trasferimento del contenuto di un file in un altro.
La parte relativa all'effettivo passaggio dati, dell'esempio illustrato in questa sezione, potrebbe essere sostituita con la seguente riga di codice:

```
// trasferisce tutti i byte del file collegato al canale di
        lettura in un altro canale
readChannel.transferTo(0, readChannel.size(),
        writeChannel);
```

FILE LOCKING

Gli argomenti finora trattati introducono delle novità nella gestione dell'I/O nel mondo Java, soprattutto in termini prestazionali e strutturali. In questa sezione verrà presentata

JAVA COMMUNITY PROCESS (JCP)

Java Community Process è un'organizzazione fondata nell'anno 1998, formata da un gruppo di esperti interno ed uno pubblico il cui compito è la guida, lo sviluppo e la supervisione delle specifiche riguardanti le tecnologie J2SE, J2EE e J2ME. Nell'area definita "expert group" si possono annoverare rappresentanti di importanti società, tra cui IBM, BEA Systems, Macromedia, Nokia, Oracle. Chiunque può entrare a far parte della community come membro pubblico e interagire attivamente al consolidamento di una data specifica attraverso commenti. Tutte le specifiche sono identificate dall'acronimo JSR (Java Specification Request) e da un numero. Il ciclo di vita completo di ogni JSR è definito da un processo, le cui regole sono descritte formalmente nel documento "JCP 2 Process Document", che comprende i seguenti step: Initiation, Community Draft, Public Draft, Maintenance. Solo le specifiche che completano tutti e quattro gli step possono essere dichiarate definitive e diventare parti potenziali della piattaforma Java. Per avere informazioni più dettagliate sul Java Community Process Program di Sun è possibile consultare il sito http://jcp.org

una funzionalità che non era attuabile in nessun modo attraverso l'uso della libreria tradizionale: il file *locking*. Questa interessante feature ci permetterà di effettuare le seguenti operazioni sui file o su parti di essi:

- **lock condiviso:** viene solitamente richiesto dai processi che devono effettuare operazioni di lettura. È ottenuto solo quando non ci sono lock di tipo esclusivo. Come si percepisce dal nome, più lock condivisi possono coesistere sul medesimo file.
- **lock esclusivo:** viene solitamente richiesto dai processi che devono effettuare operazioni di scrittura. Questo tipo di lock nega successive richieste di lock sia di tipo condiviso che esclusivo. È ottenuto solo quando il file risulta libero da ogni tipo di lock.
- **unlock:** rilascia sia i lock condivisi che esclusivi.

La caratteristica più entusiasmante di questo meccanismo è che i lock vengono impostati a livello di processo. Se, ad esempio, si imposta un lock esclusivo su un file, nessun altro processo esterno o thread interno alla JVM in esecuzione, potrà acquisire un altro lock sullo stesso file. Esistono due diverse modalità di richiedere un lock, entrambe accessibili su una reference di un oggetto *FileChannel:*

```
// modalità bloccante
FileLock blockingLock = fileChannel.lock(0, 1024, false);
// modalità non bloccante
FileLock nonBlockingLock = fileChannel.tryLock(0,
                                              1024, true);
```

Entrambe le istruzioni richiedono un lock sui primi 1024 bytes del file collegato al canale. L'ultimo argomento passato ai due metodi, quello booleano, a true richiede un lock condiviso, a *false* richiede un lock esclusivo. La principale differenza è che il primo metodo rimane bloccato fino all'effettivo ottenimento del lock, mentre il secondo riesce ad acquisire il lock solo se effetivamente disponibile al momento della richiesta. Se la richiesta di lock non viene concessa, il metodo *tryLock* ritorna *null* o solleva un'eccezione di tipo *OverlappingFileLockException*. Nel momento in cui non ci sia più bisogno di mantenere un lock su un determinato file è possibile rimuoverlo invocando il metodo *release()* sull'istanza di *FileLock* precedentemente ottenuta. La gestione del file *locking*, come quella dei file mappati in memoria, è gestita dal sistema operativo. Pertanto è utile sapere che non tutti i sistemi operativi implementano al loro interno il lock condiviso. Nel caso in cui tale funzione non sia disponibile, NIO richiede, in maniera "trasparente" al programmatore, un lock di tipo esclusivo.

I/O VS NEW I/O: READ-WRITE A CONFRONTO

Per verificare l'effettivo miglioramento delle performance sono stati effettuati dei test riguardanti la copia di file di diverse dimensioni. Sono stati utilizzati i seguenti meccanismi di I/O: tradizionale (i/o buffer), *NIO direct buffer, NIO MappedByteBuffer, NIO channels transfer mode.* La piattaforma di test utilizzata era basata su una workstation biprocessore, equipaggiata con CPU AMD Opteron da 1.99 GHz, 2GB di RAM ed una hard disk SCSI. Le prove sono state eseguite su due differenti sistemi operativi: Solaris 10 e Suse Linux 9.2. La versione di Java Virtual Machine utilizzata era la 1.5.0_03-b07.
Le tecniche che hanno fatto registrare i risultati migliori sono sottolineate con il colore di sottofondo verde. Di contro, le prestazioni peggiori sono evidenziate dal colore rosso. È importante notare che il mapping dei file in memoria risulta essere la migliore soluzione nella maggior parte dei casi, e diventa sempre più performante quando si vanno a manipolare file di grosse dimensioni. Inoltre, confrontando le due tabelle, è abbastanza intuitivo apprezzare la differenza di prestazioni tra le JVM dei due diversi sistemi operativi a parità di dispositivi hardware: l'implementazione delle operazioni di I/O di Linux risultano essere migliori di quelle di Solaris. Il lettore che volesse comparare le prestazioni del proprio sistema con quelle riportate in questa sezione può eseguire la classe NIOPerformanceTest allegata al presente articolo.

Suse Linux 9.2

	100K	1M	10M	30M
io buffer (java.io)	1.2	11.6	122.4	320.6
Nio direct buffer	11	27.6	244.2	388.8
Nio mapped byte buffer	1.4	2.8	24.2	71
Nio channel transfer mode	1	2.8	24.2	72.6

Solaris 10

	100K	1M	10M	30M
io buffer (java.io)	1	11	96	477.2
Nio direct buffer	11	23.8	197.6	575.2
Nio mapped byte buffer	2	3.4	36.4	105.4
Nio channel transfer mode	1	4.2	39	642.8

Tempi di R/W (millisecondi) con file di diverse dimensioni su OS differenti

CONCLUSIONI

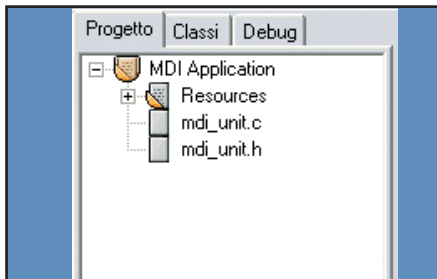
Le API NIO rappresentano un fondamentale cambiamento nella gestione dei meccanismi di Input/Output nella tecnologia Java.
Il presente articolo ha lo scopo di illustrare le basi di questo nuovo sistema insieme alle principali strutture per la gestione dei file.
È inoltre importante sapere che le novità introdotte da questa nuova libreria non si limitano solo a questo, ma integrano anche altre utili funzionalità come ad esempio il parsing attraverso *regular expressions*, la gestione asincrona delle connessioni via socket e tanto altro ancora.

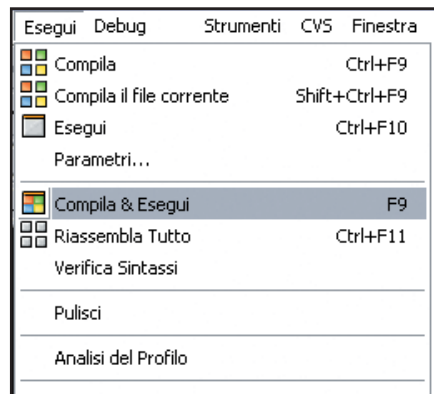
Fabrizio Fortino

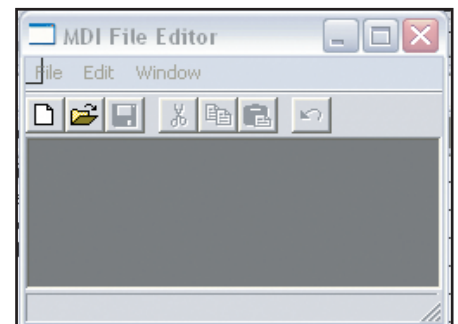
Un esempio guidato di MDI in DEV C++

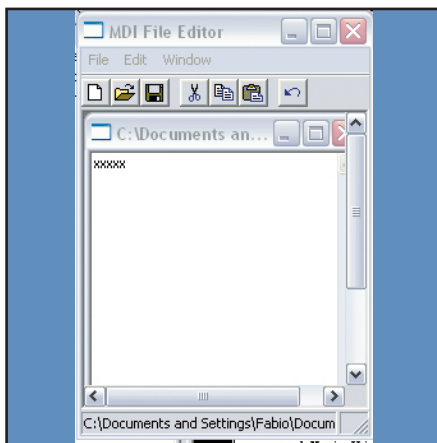
Le applicazioni in ambienti visuali comunicano con gli utenti mediante finestre. È noto come in tali elementi di comunicazioni possano trovarsi altri oggetti come: menu, bottoni, immagini e altro. Le applicazioni in questi ambiti si dividono in due grandi categorie le SDI (single document interface) e MDI (multiple document interface). Le prime sono anche conosciute come finestre di dialogo e sono semplicemente composte da uno o più oggetti ad essa associati, che non comprendono altre finestre "figlie". MDI la seconda categoria gestisce invece le finestre figlie dando la possibilità di aprirle all'interno di un'applicazione principale. In tal caso quindi, tra i vari oggetti presenti nelle finestre, possono esserci altre finestre. Ovviamente, le finestre figlie possono spostarsi solo all'interno dell'applicazione principale. Si tratta quindi di uno strumento più potente rispetto a SDI ma non sempre necessario. Diremo che il caso di usarlo ogni qual volta sorge l'esigenza di aprire finestre dentro altre finestre. Ho scovato un bel esempio allegato al compilatore DEV C++ che in modo semplice e intuitivo propone un progetto per la realizzazione e la gestione di MDI. Si tratta di un semplice editor. Per quanto ci riguarda focalizzeremo l'attenzione sulla gestione delle MDI. Seguiremo i passi per produrre l'applicazione funzionante e metteremo un po' le mani nel codice per personalizzare il risultato.

Stefano Vena

‹1› IL PROGETTO MDI APPLICATION

Dopo aver lanciato Dev C++ ci preoccupiamo di aprire un nuovo progetto. Esploriamo tra le sottocartelle di DevC++ e selezioniamo examples, tra questi sceglia-mo MDI application. Apriamo il file di progetto presente. Come si potrà notare sono presenti tre file: rispettivamente con estensione c, h e rc.

‹2› ESECUZIONE DEL PROGETTO

Verifichiamo subito il risultato. Per farlo eseguiamo in sequenza la compilazione e l'esecuzione selezionando la voce apposita dal menu esegui. In alternativa si può semplicemente digitare F9. Ancora in alternativa si può accedere dalla barra degli strumenti allo stesso bottone che esegue la funzione appena descritta.

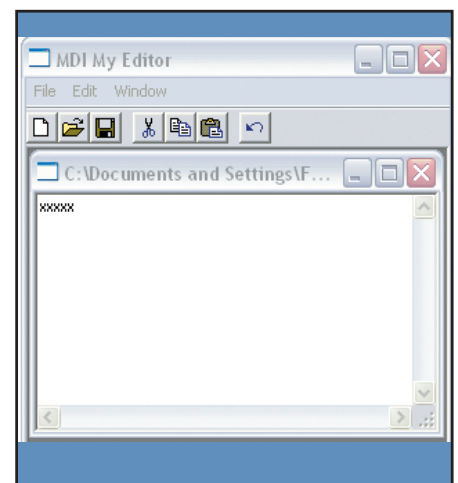
‹3› L'APPLICAZIONE MDI

Come risultato apparirà una completa applicazione di tipo MDI. Si tratta nel caso specifico del più spartano degli editor, ancora di più di notepad. All'esecuzione si apre però una sola finestra. Ma la presenza di una zona grigia all'interno della stessa ci lascia intendere che si possa riempire con un'altra finestra.

‹4› L'APPLICAZIONE "FIGLIA"

Cliccando sul bottone di nuovo, il foglietto bianco, si genera un nuovo documento vuoto. Una finestra a sfondo bianco, interna all'applicazione principale, dove poter scrivere apparirà; si tratta dell'applicazione figlia. Possiamo manipolarla come vogliamo: scriverci dentro, ridimensionare la finestra, salvare con nome e altro ancora.

‹5› PERSONALIZZIAMO IL TUTTO

```
_hMainWindow = CreateWindowEx (
        WS_EX_APPWINDOW, g_szappName,
"MDI My Editor", WS_OVERLAPPEDWINDOW |
        WS_CLIPCHILDREN,
CW_USEDEFAULT, CW_USEDEFOULT, CW_USEDEFAULT,
        CW_USEDEFOULT,
0, 0, hInstance, NULL);
```

Dando uno sguardo al codice si osserva come il file con estensione h (header), contiene parametri propri degli oggetti in gioco. Il file c contiene, invece, il codice riferito all'applicazione vera e propria. Per personalizzarlo ho cambiato la stringa "MDI file editor" in "MDI my editor". Ovviamente, esplorando tale sezione si capisce a fondo come funzionano le MDI in Dev C++.

‹6› IL NUOVO RISULTATO

Per visionare il nuovo risultato ottenuto, dopo aver salvato le modifiche approntate, ripetiamo le operazioni al punto due. Si potrà notare il nuovo titolo dell'applicazione principale. La personalizzazione così non ha praticamente limiti.

Realizzare una classe per estrarre i tokens da una stringa

Con il termine token, letteralmente gettoni, si identificano una o più parole contenute in una frase e separate da un separatore. Ad esempio, consideriamo la frase "ioProgrammo, tutti i giorni" e consideriamo come separatori la virgola, i tokens sono due "ioProgrammo" e "tutti i giorni". Ora scegliamo come separatore lo spazio, i tokens ora sono quattro e sono i seguenti: "ioProgrammo," "tutti" "i" "giorni". Una classe che ci aiuti nella separazione dei tokens potrebbe tornarci utile, ad esempio, per realizzare un piccolo parser, Infatti potrebbe semplificare la lettura di un file di configurazione scritto in stile "argomento"= "valore";. L'uguale ed il punto e virgola sono i separatori. La classe che andremo a realizzare si basa sull'utilizzo delle funzioni standard di C++ quindi è completamente portabile. Ciò di cui abbiamo bisogno come al solito è il nostro compilatore c++ preferito, un editor di testo e un po' di dimestichezza con il linguaggio. Per realizzare il seguente esempio potremmo utilizzare l'ambiente di sviluppo (freeware) Dev-C++, scaricabile gratuitamente dal sito web: *www.bloodshed.net*

Stefano Vena

<1> LE PRIME RIGHE DI CODICE

```
#ifndef TOKENS_H
#define TOKENS_H
#include <cstdlib>
#include <string>
#include <vector>
using namespace std;
class StringTokenizer
{ private:
    vector< char* > m_tokens;
```

La nostra classe è semplicemente un wrapper per le funzioni di *stringtokenizer* della libreria standard di C++ quindi le funzioni che realizzeremo sono composte da poche righe di codice. Per questo motivo la mia scelta ricade su una classe scritta completamente "inline". Creiamo un nuovo file e lo chiamiamo *"tokens.h"* poi inseriamo l'inclusione di alcuni header, dichiariamo il namespace da utilizzare ed in fine inseriamo le prime righe di codice della classe *"StringTokenizer"*. La classe conterrà solo una variabile: la lista dei tokens estratti. Ora passiamo al passaggio successivo.

<2> CREARE E DISTRUGGERE

```
public :
 StringTokenizer ( )
 {}
 ~StringTokenizer ( )
 { ClearTokens(); }
private:
void ClearTokens(){
for( vector<char* >::iterator it = m_tokens.begin();
    it != m_tokens.end(); it++ )
    { free ( *it ); }
  m_tokens.clear();
 }
```

La classe prevede un solo costruttore che praticamente non fa niente. Il distruttore in vece invoca il metodo *ClearTokens()* il quale si occupa di liberare la memoria occupata dai singoli tokens. L'iterazione attraverso gli elementi del vettore *m_tokens* viene fatta ,elegantemente, attraverso l'utilizzo di un iteratore. La memoria associata ad ogni token viene liberata attraverso il metodo *free()* poiché essa viene allocata tramite *malloc()*. Il codice inserito in questo passo non necessita altre spiegazioni.

<3> TOKENIZE, PRIMA PARTE

```
public:
 int Tokenize ( const char* stringa,
             const char* separatori)
   {
   ClearTokens();
   char * temp = strdup( stringa );
   char *token = NULL;
   token = strtok( temp, separatori );
```

Cominciamo ad analizzare la funzione principale ovvero *Tokenize*. Essa prende come parametri di input due stringhe: la stringa da suddividere e la lista dei separatori. La prima cosa da fare e svuotare l'elenco di tokens corrente. Poi è necessario creare una copia della stringa di riferimento tramite il comando *strdup()*. Questa operazione è necessaria in quanto la funzione che andremo a vedere fra poco sostituisce i separatori trovati all'interno della stringa di riferimento con il carattere *'0'* di fine stringa modificandone il valore iniziale. Ora dichiariamo la variabile token come puntatore a carattere e iniziamola con il valore *NULL;* Ora siamo pronti ad estrarre i tokens quindi invochiamo la funzione *strtok* e passiamo ad essa la stringa duplicata e la stringa con l'elenco dei separatori.

<4> TOKENIZE, PARTE 2ª

```
while( token != NULL )
{  m_tokens.push_back( strdup(token) );
    token = strtok( NULL, separatori ); }
 free(temp);
 return TokensCount(); }
```

Ora entriamo in un ciclo di *while* finche la variabile *token* non assume il valore *NULL* ovvero fino a quando non ci saranno più token da estrarre. Le operazione all'interno del *while* sono due: la prima aggiunge il duplicato del token in coda alla lista dei tokens, la seconda cerca il token successivo. La funzione *strdup()*, crea una copia della stringa che riceve come parametro allocando la memoria necessaria attraverso una chiamata ad *malloc()* è quindi necessario liberare tale risorsa attraverso il comando *free()*. La funzione *strtok()* dopo la prima invocazione, richiede che il parametro riguardante la stringa con i tokens sia passato come *NULL*.

<5> OTTENERE I TOKENS

```
int TokensCount()
{  return m_tokens.size();}
const char* GetToken( int idx )
{ if ( idx < 0 || idx >= TokensCount() )
    return "";
 return m_tokens[idx]; } };
#endif
```

A questo punto andiamo ad aggiungere le ultime due funzioni che ci permettono di ottenere i token estratti. La funzione *TokensCount* non necessita commenti, banalmente, restituisce il numero di tokens inseriti nell'elenco. La seconda funzione *"GetToken"* restituisce i token estratti oppure una stringa vuota se l'indice passato alla funzione non fa parte dell'elenco. La classe non contiene altri metodi ed è pronta per essere utilizzata.

<6> UTILIZZIAMO LA CLASSE

```
const char * stringa = "Io Programmo,
tutti i giorni";
const char * separatori = " ,";
StringTokenizer tkz;
int tokens = tkz.Tokenize( stringa ,  separatori );
for( int i = 0; i < tokens; i++ )
{
  printf( "Token %d =  \"%s\" \n",i+1,
  tkz.GetToken(i) );
}
```

Questa è una carrellata delle potenti e utili funzionalità della classe appena creata.
Buon utilizzo.

Compilare ed utilizzare sqlite

Ognuno di noi, almeno una volta, nella propria carriera di programmatore, si è imbattuto in soluzioni che necessitassero dell'utilizzo di un database. A questo punto le soluzioni sono molteplici ed ognuna di esse presenta vantaggi e svantaggi. In questo express andremo a vedere come compilare le librerie di sqlite e come integrare il famoso motore di database all'interno delle nostre applicazioni c++. Sql Lite è un SQL Database Engine che si integra completamente con le applicazioni che ne fanno uso in quanto risiede tutto in unica libreria. La libreria può essere importata sia dinamicamente che staticamente. Alcuni degli svantaggi di tale soluzione sono imputabili all'assenza di alcune funzionalità avanzate. Tutte le informazioni inerenti la libreria e il codice sorgente sono disponibile al sito internet *http://www.sqlite.org/*. Ciò di cui abbiamo bisogno, per realizzare la libreria ed importarla in c++, come al solito è il nostro compilatore c++ preferito, un editor di testo e un po' di dimestichezza con il linguaggio. In questo esempio abbiamo utilizzato il compilatore MINGW32 e l'emulatore di shell Msys. Di conseguenza l'intero express può essere riprodotto sotto ambiente linux.

Stefano Vena

<1> COMPILARE LA LIBRERIA

```
tar xzf sqlite-3.2.2.tar.gz
mkdir build
cd build
../sqlite-3.2.2/configure
make
```

La prima cosa da fare è, ovviamente, scaricare il file *sqlite-3.2.2.tar.gz* dalla sezione download del sito *http://www.sqlite.org/*.
Una volta scaricato il file attraverso la shell raggiungiamo il percorso dove si trova il pacchetto con i sorgenti ed eseguiamo la sequenza di comandi riportata in alto.
La libreria verrà generata nella directory build/.libs con il nome *libsqlite3.a.*
A questo punto copiamo tale file nella directory lib di *MINGW32.*
Nella directory include del compilatore andremo a copiare il file *sqlite3.h* situato in build.
Sotto linux possiamo semplicemete utilizzare il comando *make install.*

<2> SQL LITE IN C++: 1ª PARTE

```
#include <stdio.h>
#include "sqlite3.h"

using namespace std;
static int showq(void *NotUsed, int argc, char **argv,
                                          char **azColName)
{  int i;
   for(i=0; i<argc; i++){
   printf("%s = %s\n",
   azColName[i],
   argv[i] ? argv[i] : "NULL"); }
  printf("\n");
  return 0; }
static bool ParseErrors( const int &rc, const char* Msg,
                    sqlite3 *db){ if( rc!=SQLITE_OK ){
              fprintf(stderr,"Errore SQL : %s\n",Msg);
 sqlite3_close(db);
 return true;
 } else return false; }
```

Ora siamo pronti ad utilizzare la libreria appena creata nei nostri programmi. La prima cosa da fare e includere alcuni header. Oltre al classico header stdio.h includiamo il file sqlite3.h. La prima funzione ad essere inserita è una funzione di callback che utilizzeremo in seguito per visualizzare i risultati delle query sql effettuate sul database. La seconda verrà utilizzata per processare gli eventuali errori, quindi liberare le risorse occupate dal motore.

<3> SQL LITE IN C++. 2ª PARTE

```
int main(int argc, char *argv[])
{
  sqlite3 *db;
  char *Msg = 0;
  int rc;

  remove( "database.db" );
  rc = sqlite3_open("database.db", &db);

  if( ParseErrors(
rc, sqlite3_errmsg(db), db) ) return -1;
```

Ora scriviamo un piccolo main in grado di creare il database, le Table e di accedere ad esse. Per prima andiamo eliminiamo , se esiste, un database precedente e poi andiamone a creare uno nuovo. Il metodo *sqlite3_open()* si occupa di aprire il database indicato dalla stringa se esso esiste oppure crearne uno nuovo. Succesivamente, se avvengono degli errori, la funzione *ParseErrors()* ritorna true quindi usciamo dal programma altrimenti proseguiamo.

<4> SQL LITE IN C++. 3ª PARTE

```
const char* create = "create table Esempio("
"Id int, "
"Nome char(50),"
"Cognome char(50) );";

const char* insert0 = "insert into Esempio "
"values (0, 'Mario', 'Rossi');";

const char* insert1 = "insert into Esempio "
"values (1, 'Marco', 'Verde');";

const char* query =
"select * from Esempio order by id asc;";
```

Con la classica sintassi sql andiamo a definire quattro query che serviranno, rispettivamente, per creare una tabella *Esempio*, inserire un primo elemento,inserire un secondo elemento ed infine realizzare un select sul database.

<5> LA COMPILAZIONE

```
g++ main.cpp  -o "SQLite.exe" -lsqlite3
```

Siamo quasi pronti per compilare e vedere il risultato dei nostri sforzi.
La fase di compilazione è davvero banale.
Sia sotto la shell di linux o in generale di un sistema unix-like che tramite Msys sotto windows la sintassi è identica.
L'unica accortezza da usare è di inserire correttamente nei percorsi di consultazione del compilatore i files "libsqlite3.a" e "sqlite3.h"

<6> CONCLUSIONI

```
rc= sqlite3_exec(db,create, 0, 0, &Msg);
if( ParseErrors( rc, Msg, db)) return -1;
rc= sqlite3_exec(db,insert0, 0, 0, &Msg);
if( ParseErrors( rc, Msg, db)) return -1;
rc= sqlite3_exec(db,insert1,0,0, &Msg);
if( ParseErrors( rc, Msg, db)) return -1;
rc=sqlite3_exec(db,query,showq,0,&Msg)
if( ParseErrors( rc, Msg, db) )return -1;
sqlite3_close(db);
return 1;
}
```

Le chiamate fondamentali sono ovviamente quelle riportate nello spezzone qui riportato.
Il codice in questione non richiede particolari commenti. Vengono eseguite le varie query ed in caso di errori usciamo dal programma. Da notare che la funzione di *callback* viene utilizzata solo nell'ultimo exec dove è necessaria la stampa dei dati.

Instant Developer il Web è servito!

Cosa è "Instant Developer?". Non esiste una risposta definitiva a questa domanda, di fatto "Instant Developer" apre la strada ad una nuova tipologia di software di sviluppo per "Web Application"

Instant Developer si avvicina a quello che normalmente viene definito come ambiente RAD ereditandone la capacità di creare applicazioni basate su componenti, sul drag & drop, e sulla modalità di "disegno dell'applicazione" tipica degli ambienti RAD. Pro Gamma lo definisce "il primo sistema di sviluppo relazionale al mondo" perché è in grado di mantenere automaticamente integrate tutti le parti dell'applicazione, cosa che è a carico dei programmatori nei tradizionali ambienti.

UN CASO DI STUDIO

Un vostro cliente fino ha ieri ha gestito il magazzino della sua azienda tramite un database access, essendo notevolmente cresciuto vuole passare a una gestione più professionale della propria attività. Vi chiede di realizzare un'applicazione tale che sia accessibile ovunque sia disponibile una connessione Internet, di modo che gli agenti possano effettuare ordini e visualizzare il contenuto del magazzino. Desidera anche che anche gli impiegati in sede possano accedere all'applicazione per ad esempio verificare il livello di approvvigionamento delle scorte ed eventualmente provvedere a rifornire il magazzino. Infine con un livello di accesso differente desidera ottenere dei report dell'attività da poter utilizzare per effettuare le proprie scelte di marketing. Per realizzare il tutto in tempi rapidi dovreste: scegliere una tecnologia Windows/Unix, scegliere un linguaggio fra tutti quelli disponibili, scrivere diverse migliaia di

INFORMAZIONI COMMERCIALI

Il prodotto in allegato a ioProgrammo è completo in versione demo per 30 giorni. Terminati i 30 giorni di valutazione i lettori di ioProgrammo possono ottenere gratuitamente una licenza *Learning Edition* del valore di 500 euro.

La *Learning Edition* differisce dalla versione completa limitatamente al numero di tabelle utilizzabili in un progetto e al numero di documenti /videate, rispettivamente fissati in 10 tabelle e 20 documenti.
Per ottenere la licenza Learning Edition è necessario collegarsi all'indirizzo http://www.instantdeveloper.com/learning/registrazione.asp, e inserire la Key fornita dal software al momento del lancio in congiunzione al codice promozionale IOP2230LE e i propri dati personali corretti.

REQUISITI

Conoscenze richieste
Basi di programmazione

Software
Apache Tomcat o IIS, Apache Ant, J2SE 1.5. Trovate tutto eccetto IIS nel CDRom allegato a ioProgrammo

Impegno

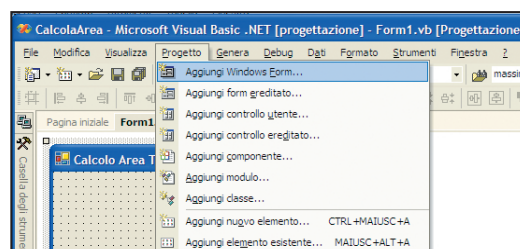
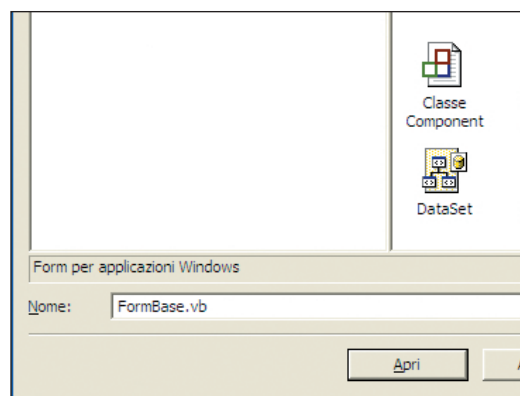
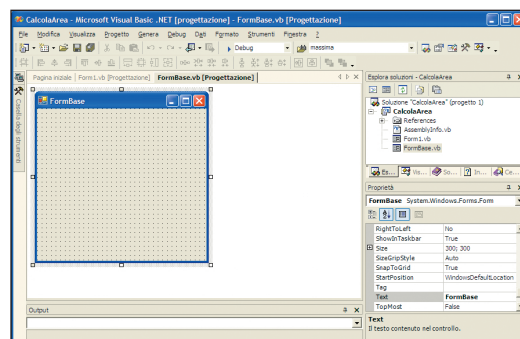
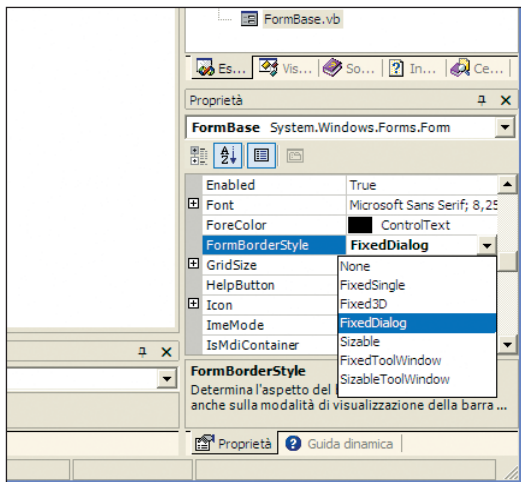
Tempo di realizzazione

Dialoghiamo con le finestre

Le finestre sono, o quasi, l'unico metodo per garantire l'interazione fra utente e applicazione. Impariamo come usarle da Visual Basic, quali sono i vari tipi supportati e come personalizzarle

Le finestre di dialogo costituiscono buona parte dell'interfaccia utente, nelle applicazioni Windows. Vengono utilizzate per mostrare dei messaggi, oppure per richiedere all'utente dati necessari all'applicazione. VB.NET 2003 fornisce molte finestre di dialogo standard che è possibile adattare alle applicazioni, ad esempio la finestra *Apri*, rappresentata dal controllo *OpenFileDialog*, oppure la finestra di *Stampa* rappresentata dal controllo *PrintDialog*. In VB.NET è possibile, inoltre, progettare finestre di dialogo completamente personalizzate in base ad esigenze particolari. Una finestra di dialogo, in pratica, non è altro che una form modale con un insieme di controlli (tipicamente *Label*, *Textbox* e *Button*), la cui proprietà *FormBorderStyle* è impostata su *FixedDialog*.

CREARE UNA FINESTRA DI DIALOGO DA ZERO

Descriviamo subito come realizzare delle finestre di dialogo personalizzate, creando un nuovo progetto *Windows Applications* dal nome *CalcolaArea*. L'applicazione che andremo a realizzare, dovrà semplicemente calcolare l'area di un triangolo con le dimensioni di base ed altezza inserite dall'utente. A differenza dell'applicazione *CalcolaArea* vista in un numero precedente, la finestra iniziale includerà soltanto un controllo *Button* dal nome *ButtonCalcola*. Quando l'utente cliccherà sul pulsante verrà mostrata una prima finestra di dialogo in cui verrà chiesto di inserire il valore della base del triangolo, una seconda finestra di dialogo in cui verrà chiesto di inserire il valore dell'altezza del triangolo, ed infine una terza finestra di dialogo che mostrerà il risultato del calcolo.
Per creare la prima finestra di dialogo dobbiamo aggiungere una form al progetto e modificarne alcune proprietà, vediamo come:

1 Selezioniamo la voce: *Progetto'Aggiungi Windows Form*

2 Digitiamo il nome della finestra *(FormBase)* nella casella di testo *Nome*

3 Clicchiamo sul pulsante *Apri*, in questo modo verrà mostrata la nuova form dal nome *FormBase*, precedentemente inserito

4 Dalla finestra delle proprietà, selezioniamo la proprietà *FormBorderStyle* e poniamola a *FixedDialog*. Selezioniamo le proprietà *MinimizeBox* e *MaximizeBox* e poniamole a *False*, poiché le finestre di dialogo in genere non includono pulsanti di riduzione ad icona e di ingrandimento.

Le finestre di dialogo vengono visualizzate come finestre modali, e cioè, come finestre a scelta obbligatoria, che impediscono all'utente di eseguire operazioni al di fuori della stessa finestra di dialogo. Per visualizzare una finestra come modale, si deve utilizzare il metodo *ShowDialog*, pertanto possiamo scrivere il codice seguente:

```
Private Sub ButtonCalcola_Click(ByVal sender As
          System.Object, ByVal e As System.EventArgs)
                              Handles ButtonCalcola.Click
     Dim FBase As New FormBase()
     FBase.ShowDialog()
End Sub
```

La finestra *FormBase* viene utilizzata per richiedere all'utente di inserire i dati relativi alla misura della base del triangolo. Nel nostro caso diventa importante sapere in che modo viene chiusa la finestra, in altre parole se ha prodotto un risultato oppure se, l'utente ha chiuso la finestra dalla crocetta in alto a destra eliminando i dati inseriti. Per verificare come viene chiusa una finestra di dialogo, possiamo utilizzare la proprietà *DialogResult*. In fase di progettazione è possibile impostare la proprietà *DialogResult* per tutti i controlli *Button* presenti nella finestra di dialogo.

COSTRUIAMO LA FINESTRA FORMBASE

- Selezioniamo la finestra *FormBase* e, se non è già visualizzata, apriamo la finestra delle proprietà: selezioniamo la proprietà *Text* e modifichiamo il testo visualizzato nella barra del titolo in: *Inserisci la misura della Base del Triangolo.*
- Selezioniamo un controllo *TextBox* dalla casella degli strumenti (nella sezione *Windows Form*) e disegniamo il controllo sulla form.
- Selezioniamo il *Textbox* e, dalla finestra delle proprietà, evidenziamo la proprietà *Name* per cambiare il nome in: *TextBoxBase*. Evidenziamo la proprietà *Text* e cambiamo il testo nella stringa vuota
- Selezioniamo un controllo *Label* dalla casella degli strumenti, e disegniamo il controllo sulla form in corrispondenza del *TextBox* disegnato in precedenza.
- Selezioniamo la *Label* e, dalla finestra delle proprietà, evidenziamo la proprietà *Text* per cambiare il testo visualizzato in: *Base*.
- Selezioniamo per due volte un controllo *Button* dalla casella degli strumenti e disegniamo i due controlli sulla form ai lati del *TextBox*
- Selezioniamo il primo *Button* e, dalla finestra delle proprietà, evidenziamo la proprietà *Name* per cambiare il nome in: *ButtonOk*. Evidenziamo la proprietà *Text* e cambiamo il testo in: *OK*. Evidenziamo la proprietà *DialogResult* e cambiamo il valore in: *OK*.
- Allo stesso modo selezioniamo il secondo *Button* e variamo la proprietà *Name* in: *ButtonCancella*, la proprietà *Text* in: *Cancella* e la proprietà *DialogResult* in *Cancel*

Dalla finestra (nel nostro caso *FormCalcola*) che visualizza la finestra di dialogo, chiamata anche form padre della finestra di dialogo, possiamo utilizzare il valore della proprietà *DialogResult* per stabilire se è stato scelto il comando *OK* o *Annulla*. Possiamo scrivere ad esempio:

```
Dim Base As Double
If FBase.DialogResult = DialogResult.OK Then
     Base = CDbl(FBase.TextBoxBase.Text)
Else
     Base = 0
End If
```

A questo punto dobbiamo costruire la finestra *FormAltezza* seguendo le stesse operazioni compiu-

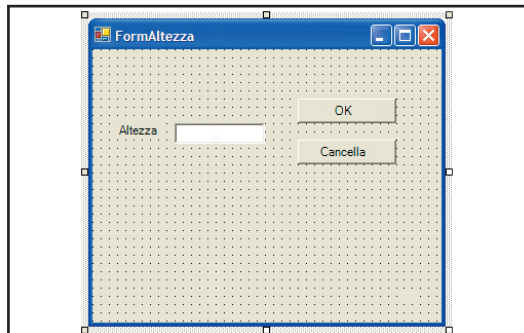
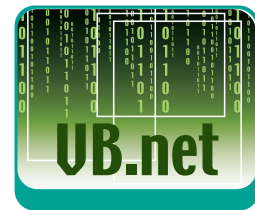



Fig. 1: ***La finestra di dialogo FormAltezza***

Il controllo *FolderBrowserDialog* permette di visualizzare un'interfaccia utilizzabile per sfogliare e selezionare una cartella o crearne una nuova.

NOTA

Il controllo *PageSetupDialog* permette di visualizzare la finestra di dialogo *Imposta pagina* che consente all'utente di stabilire l'orientamento della pagina (verticale o orizzontale), le dimensioni del foglio e le dimensioni per i quattro margini della pagina. Anche in questo caso, prima di utilizzare il controllo *PageSetupDialog,* è necessario inserire nella form il controllo *PrintDocument* ed associarlo al controllo *PageSetupDialog* tramite la proprietà *Document.*

te per la finestra *FormBase*. Per concludere, il risultato dell'operazione di calcolo dell'area del triangolo, sarà visualizzato in una finestra di dialogo predefinita, la finestra *MessageBox*. Di seguito il codice necessario per il corretto funzionamento dell'applicazione:

```
Private Sub ButtonCalcola_Click(ByVal sender As
        System.Object, ByVal e As System.EventArgs)
                    Handles ButtonCalcola.Click
    Dim Base As Double
    Dim Altezza As Double
    Dim Risultato As Double
    Dim FBase As New FormBase
    FBase.ShowDialog()
    If FBase.DialogResult = DialogResult.OK Then
        Base = CDbl(FBase.TextBoxBase.Text)
    Else
        Base = 0
    End If
    Dim FAltezza As New FormAltezza
    FAltezza.ShowDialog()
    If FAltezza.DialogResult = DialogResult.OK Then
        Altezza = CDbl(FAltezza.TextBoxAltezza.Text)
    Else
        Altezza = 0
    End If
    Risultato = (Base * Altezza) / 2
    MessageBox.Show(risultato)
End Sub
```

LA FINESTRA DI DIALOGO MESSAGEBOX

La finestra di dialogo *MessageBox*, è senza dubbio la finestra di dialogo che ci troveremo ad utilizzare più spesso. Questa finestra permette di mostrare messaggi personalizzati all'utente, ed è in grado di accettare scelte tramite uno o più pulsanti.
Normalmente è composta da quattro parti.

1. La *Barra* del titolo che identifica lo scopo della finestra di dialogo, ad esempio *"Richiesta Conferma"*
2. Il messaggio che appare nella finestra, ad esempio *"Sei sicuro di voler continuare?"*
3. Un'icona in grado di attirare l'attenzione, ad esempio l'icona con il punto di domanda
4. Uno o più pulsanti di comando, ad esempio i pulsanti *Si* e *No*

Per visualizzare la finestra di messaggio, dobbiamo utilizzare il metodo *Show.* Il metodo *Show* può essere richiamato in più modi, tra quelli più comuni:

```
MessageBox.Show(testo)
MessageBox.Show(testo, etichetta)
MessageBox.Show(testo, etichetta, bottone, icona)
MessageBox.Show(testo, etichetta, bottone, icona,
                                  bottoneDiDefault)
```

In cui:

- **testo** - rappresenta il messaggio che verrà visualizzato all'interno della finestra di dialogo. Può essere una qualsiasi stringa ed è l'unico parametro obbligatorio.
- **etichetta** - rappresenta la stringa visualizzata nella barra del titolo della finestra. Se viene omesso, nella barra del titolo non apparirà nessun testo.
- **bottone** - permette di visualizzare uno o più pulsanti, tra quelli disponibili, nella finestra di dialogo. Se viene omesso, verrà visualizzato il bottone *OK.*
- **icona** - permette di visualizzare un'icona, tra quelle disponibili, nella finestra di dialogo. Se viene omesso, non verrà visualizzata nessun'icona.
- **BottoneDiDefault** - permette di specificare quale pulsante, tra quelli visualizzati, verrà impostato come predefinito. L'uso di questo parametro, consente all'utente di leggere il messaggio e di premere il tasto *Invio* per indicare l'azione del pulsante predefinito.

Vb.Net mette a disposizione un set limitato di possibili valori per *bottone, icona* e *BottoneDiDefault.* In particolare, *bottone* può assumere i seguenti valori:

- **AbortRetryIgnore** - Indica che la finestra di messaggio contiene i pulsanti *Interrompi, Riprova* ed *Ignora.*
- **OK** - Indica che la finestra di messaggio contiene il pulsante *OK* .
- **OKCancel** - Indica che la finestra di messaggio contiene i pulsanti *OK* ed *Annulla.*
- **RetryCancel** - Indica che la finestra di messaggio contiene i pulsanti *Riprova* ed *Annulla.*
- **YesNo** - Indica che la finestra di messaggio contiene i pulsanti *Sì* e *No.*
- **YesNoCancel** - Indica che la finestra di messaggio contiene i pulsanti *Sì, No* ed *Annulla.*

Quando l'utente preme uno dei tasti riportati nella finestra di dialogo, viene valorizzata la proprietà *DialogResult* corrispondente. Le icone che si possono visualizzare nella finestra *MessageBox* sono:

- **Asterisk, Information** - consentono di visualizzare un'icona contenente un simbolo formato da una lettera *"i"* minuscola racchiusa da un fumetto.
- **Error, Hand, Stop** - consentono di visualizzare un'icona contenente un simbolo formato

da una *X* bianca racchiusa da un cerchio su sfondo rosso.

- **Exclamation, Warning -** consentono di visualizzare un'icona contenente un simbolo formato da un punto esclamativo racchiuso da un triangolo su sfondo giallo.
- **Question** - consente di visualizzare un'icona contenente un simbolo formato da un punto interrogativo racchiuso da un fumetto.

Infine, non ci resta che elencare i valori utilizzabili per indicare il pulsante predefinito di una finestra *MessageBox:*

- **DefaultButton1** - indica che il primo pulsante nella finestra messaggio deve essere il pulsante predefinito
- **DefaultButton2** - indica che il secondo pulsante nella finestra messaggio deve essere il pulsante predefinito
- **DefaultButton3** - indica che il terzo pulsante nella finestra messaggio deve essere il pulsante predefinito

Il pulsante predefinito prescelto viene valorizzato in ordine da sinistra verso destra. Se abbiamo visualizzato i pulsanti *AbortRetryIgnore* ed impostato il valore di *BottoneDiDefault* su *DefaultButton3,* allora il pulsante predefinito sarà il pulsante *Ignora.*

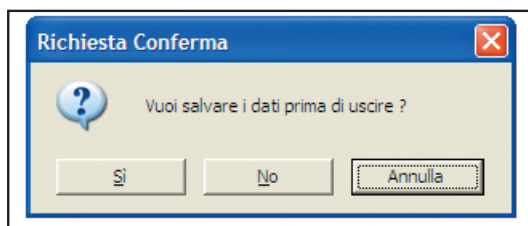
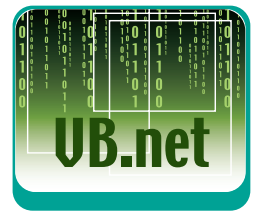



Fig. 2: ***Un classico esempio di messagebox***

Supponiamo ora di aver scritto un'applicazione che salva alcuni dati su database, se l'utente distratto chiude il programma prima di aver salvato effettivamente i dati è bene avvisarlo. In questo caso possiamo scrivere il codice seguente che visualizza il messaggio di avviso riportato in figura:

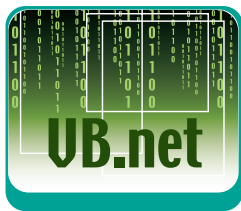
```
If MessageBox.Show("Vuoi salvare i dati prima di
                   uscire ?", _ "Richiesta Conferma",
                   MessageBoxButtons.YesNoCancel, _
                   MessageBoxIcon.Question,
                   MessageBoxDefaultButton.Button3) =
                   DialogResult.No _
Then Exit Sub
```

I CONTROLLI "FINESTRE DI DIALOGO"

VB.NET mette a disposizione alcuni controlli che permettono di visualizzare finestre di dialogo standard. Come di consueto, per utilizzare un controllo finestra di dialogo all'interno delle nostre applicazioni, si deve selezionare il controllo prescelto nella casella degli strumenti e trascinarlo sulla form.

I controlli finestre di dialogo a disposizione, sono:

- **OpenFileDialog -** permette di aprire un file.
- **SaveFileDialog -** permette di selezionare un file da salvare e la posizione in cui deve essere salvato.
- **ColorDialog** - permette di selezionare o aggiungere un colore da una tavolozza predefinita.
- **FontDialog -** permette di selezionare un tipo di carattere tra quelli installati nel sistema.
- **PrintDialog -** permette di selezionare una stampante e le pagine da stampare.
- **PageSetupDialog** - permette di impostare i dettagli di una pagina per la stampa.

LE PROPRIETÀ DI OPENFILEDIALOG

AddExtension	Indica se viene aggiunto automaticamente un'estensione ad un nome di file quando l'utente non la inserisce.
CheckFileExists	Indica se nella finestra di dialogo viene visualizzato un avviso quando l'utente specifica un nome di file inesistente.
CheckPathExists	Indica se nella finestra di dialogo viene visualizzato un avviso quando l'utente specifica un percorso inesistente
DefaultExt	Indica l'estensione del file predefinita
DereferenceLinks	Indica se la finestra di dialogo restituisce la posizione del file a cui fa riferimento il collegamento o se restituisce la posizione del collegamento.
FileName	Contiene il nome del file selezionato nella finestra di dialogo. È una proprietà a sola lettura
FileNames	Contiene i nomi di tutti i file selezionati nella finestra di dialogo. È una proprietà a sola lettura
Filter	Indica la stringa filtro del nome file corrente, che stabilisce le opzioni visualizzate nelle casella relative al tipo di file nella finestra di dialogo.
FilterIndex	Indica l'indice del filtro attualmente selezionato nella finestra di dialogo.
InitialDirectory	Indica la directory iniziale visualizzata dalla finestra di dialogo.
Multiselect	Ottiene o imposta un valore che indica se la finestra di dialogo consente la selezione multipla di file.
ReadOnlyChecked	Ottiene o imposta un valore che indica se è selezionata la casella di controllo di sola lettura.
RestoreDirectory	Indica se la finestra di dialogo ripristina la directory corrente prima della chiusura.
ShowHelp	Indica se viene visualizzato il pulsante di help nella finestra di dialogo.
ShowReadOnly	Indica se la finestra di dialogo contiene una casella di controllo di sola lettura.
Title	indica il titolo della finestra di dialogo che viene visualizzato nella barra del titolo.

Per visualizzare una finestra di dialogo standard, si deve utilizzare il metodo *ShowDialog*, così come abbiamo visto in precedenza. Ad esempio nel caso di un controllo *OpenFileDialog* (dal nome *OpenFileDialog1*) si deve scrivere

```
OpenFileDialog1.ShowDialog()
```

Anche le finestre di dialogo standard restituiscono la proprietà *DialogResult*. Ad esempio nel caso di un controllo *OpenFileDialog*, la proprietà *DialogResult* restituisce i valori *Ok* o *Cancel* che corrispondono, rispettivamente, ai pulsanti *Apri* e *Annulla* della finestra di dialogo. L'utilizzo di questi controlli evita la necessità di scrivere codice per implementare il disegno dell'interfaccia, ma resta sempre a carico del programmatore la scrittura del codice necessario a gestire le informazioni selezionate dall'utente.

IL CONTROLLO OPENFILEDIALOG

La finestra di dialogo *Apri* permette agli utenti di navigare tra le cartelle del proprio computer locale, o di qualsiasi computer in rete, e di selezionare un file da aprire. La finestra di dialogo restituisce il percorso completo ed il nome del file selezionato nella finestra. Il controllo non apre e legge un file, ma è semplicemente un'interfaccia che permette ad un utente di individuare e specificare il file che l'applicazione deve aprire. Per questo motivo è necessario scrivere il codice necessario alle gestione dei file, che vedremo in uno dei prossimi articoli. È possibile utilizzare la proprietà *FilterIndex* per impostare l'estensione dei file che sarà possibile visualizzare, ad esempio soltanto i file testo con estensione *.TXT*. La stringa che definisce il filtro è composta da due elementi separati da una barra verticale (per intenderci il simbolo che si trova di solito di fianco al tasto *"1"*), l'etichetta che appare nella casella di riepilogo

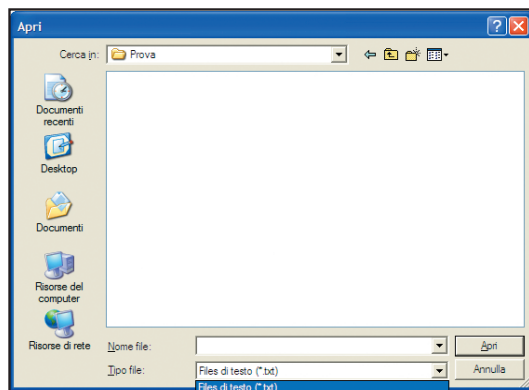



Fig. 3: **La finestra prodotta dal controllo "Openfiledialog"**

Tipo File ed il filtro stesso. Se, ad esempio, vogliamo permettere la visualizzazione dei file di tipo testo con estensione **.txt*, dobbiamo scrivere:

```
Files di testo (*.txt)|*.txt
```

Si possono anche definire dei filtri multipli, in questo caso ogni filtro deve essere separato dalla barra verticale

```
Files di testo (*.txt)|*.txt|Tutti i files (*.*)|*.*
```

In caso di filtri multipli, per definire il filtro visualizzato di default, si deve impostare la proprietà *FilterIndex* ponendola uguale all'indice numerico del filtro desiderato, nel caso precedente si deve impostare *FilterIndex* pari ad 1 oppure a 2. Per stabilire quale file è stato selezionato dall'utente si deve utilizzare la proprietà *FileName*, che memorizza il nome del file, preceduto dal percorso completo in cui si trova. Se l'utente clicca sul tasto *Annulla,* la proprietà *FileName* sarà pari alla stringa vuota ""

IL CONTROLLO SAVEFILEDIALOG

La finestra di dialogo *Salva* con *Nome* assomiglia molto alla finestra *Apri*. Le uniche differenze sono nelle etichette e nella barra del titolo, dove al posto del testo *"Apri"* viene visualizzato il testo *"Salva"*. Anche per questo controllo è possibile definire dei filtri sull'estensione del file da salvare con le stesse modalità del controllo *OpenFileDialog*. Le uniche differenze consistono nella presenza di ulteriori proprietà come, ad esempio, la proprietà *CreatePrompt* o *OverwritePrompt* che poste a *True* visualizzano un messaggio nel caso venga creato un nuovo file o si sovrascriva un file esistente.

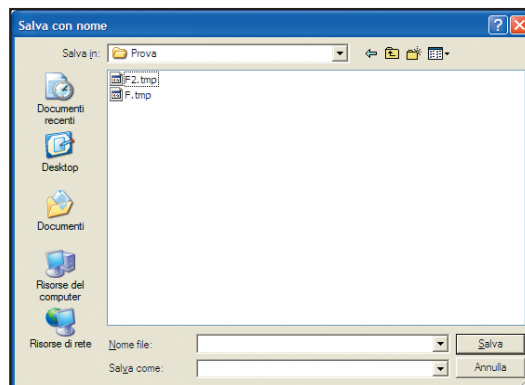



Fig. 4: **La finestra prodotta dal controllo "SaveFileDialog"**

Naturalmente il controllo non salva fisicamente un file su disco, ma è necessario scrivere del codice che lo permetta.

NOTA

PROPRIETÀ DI COLORDIALOG

AllowFullOpen: Indica se l'utente può utilizzare la finestra di dialogo per definire colori personalizzati.

AnyColor: Indica se nella finestra di dialogo sono visualizzati tutti i colori disponibili nel set di colori di base.

Color: Consente di ottenere o impostare il colore selezionato dall'utente.

CustomColors: Indica il gruppo di colori personalizzati visualizzato nella finestra di dialogo.

FullOpen: Indica se i controlli utilizzati per creare colori personalizzati sono visualizzati all'apertura della finestra di dialogo.

SolidColorOnly: Indica se nella finestra di dialogo la scelta sarà consentita soltanto ai soli colori in tinta unita

IL CONTROLLO COLORDIALOG

Il controllo *ColorDialog* permette di visualizzare la finestra di dialogo *Colore*. Questa finestra di dialogo, consente all'utente di selezionare un colore o di creare colori personalizzati. Quando l'utente chiude la casella di dialogo, il colore selezionato viene memorizzato nella proprietà *Color*. Ad esempio si può assegnare il colore selezionato alla proprietà *ForeColor* o *BackColor* di un altro controllo

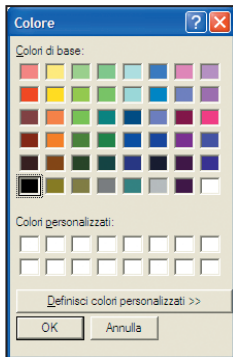



Fig. 5: ***La finestra prodotta da "ColorDialog"***

```
TextBox1.ForeColor = ColorDialog1.Color
```

La matrice *CustomColor* contiene tutti i colori personalizzati definiti dall'utente.

IL CONTROLLO FONTDIALOG

Il controllo *FontDialog* permette di visualizzare la finestra di dialogo *Carattere*. Questa finestra di dialogo, consente all'utente di selezionare diversi caratteri, stili e dimensioni. Ogni volta che l'utente seleziona un'opzione, viene mostrata una casella di anteprima con il carattere corrispondente alle scelte effettuate. Per recuperare le selezioni effettuate dall'utente, il controllo *FontDialog* espone le proprietà:

- **MinSize** - Permette di ottenere o impostare la dimensione minima selezionabile da un utente, espressa in punti.
- **MaxSize** - Permette di ottenere o impostare la dimensione massima selezionabile da un utente, espressa in punti.
- **FontMustExist** - indica se nella finestra di dialogo viene descritta una condizione di errore quando l'utente cerca di selezionare un tipo di carattere o uno stile inesistente.
- **ShowEffects** - Permette di ottenere o impostare un valore che indica se nella finestra di dialogo sono inclusi controlli che consentono all'utente di specificare opzioni di testo quali il barrato, la sottolineatura e il colore.
- **AllowVerticalFonts** - Permette di ottenere o impostare un valore che indica se nella finestra di dialogo sono visualizzati sia tipi di carattere verticali sia orizzontali oppure soltanto orizzontali.
- **AllowVectorFonts** - Permette di ottenere o impostare un valore che indica se la finestra di dialogo consente di selezionare tipi di carattere vettoriali.

Ed infine la proprietà *Font*, che imposta o restituisce il font selezionato

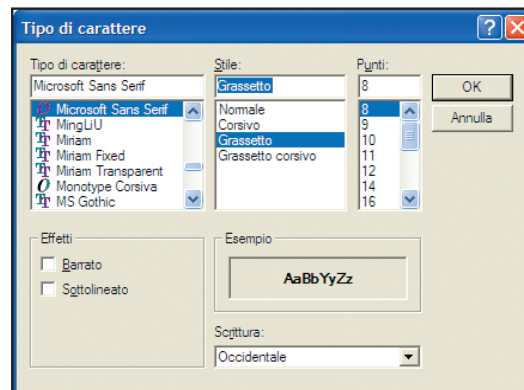



Fig. 6: ***La finestra prodotta dal controllo "FontDialog"***

IL CONTROLLO PRINTDIALOG

Il controllo *PrintDialog* permette di visualizzare la finestra di dialogo *Stampa*. Questa finestra di dialogo, consente all'utente di selezionare la stampante, il numero di copie e le pagine da stampare prima di utilizzare il controllo *PrintDialog*, è necessario inserire nella form il controllo *PrintDocument* ed associarlo al controllo *PrintDialog* tramite la proprietà *Document* (cliccando sulla freccia rivolta verso il basso accanto alla proprietà apparirà l'elenco degli oggetti *PrintDocument* selezionabili).
L'oggetto *PrintDocument* espone la proprietà *DocumentName* che dice alla finestra di dialogo *Stampa* quale documento deve essere stampato. A questo punto, in realtà, la finestra di dialogo *Stampa* non produce nessun effetto, pertanto rimane sempre a carico del programmatore la scrittura del codice necessario a stampare un documento o una parte selezionata di quest'ultimo.

Luigi Buono

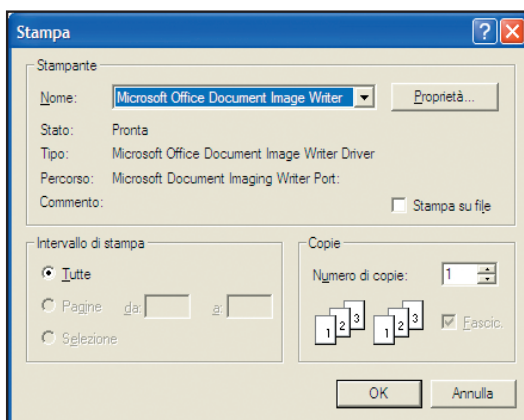
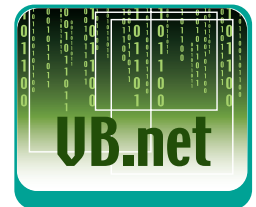



Fig. 7: ***La finestra prodotta dal controllo "PrintDialog"***

NOTA

Quando viene visualizzato una form a scelta obbligatoria, il codice successivo alla chiamata del metodo *Showdialog* non viene eseguito fino a quando la finestra visualizzata non viene chiusa, e l'input da tastiera o dal mouse è valido solo per gli oggetti del form. Viceversa nel caso di finestre modeless il codice successivo alla chiamata del metodo show viene eseguito progressivamente, la visualizzazione della nuova form non interrompe, quindi, il flusso di esecuzione del codice e l'utente può passare da questa a qualsiasi altra form dell'applicazione.

Creare controlli personalizzati

L'ereditarietà sta alla base della programmazione ad Oggetti. Poter riutilizzare il codice derivando di volta in volta classi sempre più specializzate consente di ridurre il tempo di sviluppo. Vediamo come

Uno dei principi cardini della programmazione ad oggetti è l'ereditarietà. Essa è quel meccanismo che, data una classe base, consente di derivarne una seconda che eredita dalla prima tutti i comportamenti ma che può essere estesa o modificata aggiungendone dei nuovi.
L'esempio classico è quello delle macchine:

```
public class MacchineDiesel : Macchine
{
        protected override void Accensione()
        {
                [...]
        }
}
```

Nell'esempio di cui sopra, la classe *MacchineDiesel* eredita tutti i comportamenti della classe *Macchine* e sovrascrive il metodo *Accensione* implementando strutture proprie delle macchine Diesel. All'interno di ASP.NET ogni singolo elemento, anche un banale tag HTML è convertito in automatico dal parser nella corrispettiva istanza di un control, ovvero di un oggetto derivato dalla classe *Control*. Per tale motivo ogni elemento della pagina può essere "pilotato" in modo programmatico. In buona sostanza, tutti gli elementi di una pagina sono oggetti e come tali possono essere modificati, riadattati, smontati e rimontati a nostro piacimento.

UN CONTROL È SEMPRE UN CONTROL

Non si tratta di un errore di battitura. Ogni control che trovate nella pagina eredita dalla classe *Control*, contenuta nel namespace *System.Web.UI*, tale classe fornisce i metodi di base ai controlli derivati, primo tra tutti la possibilità di effettuare il rendering delle informazioni, ovvero di mostrare a video dei contenuti.
Ora, il perché il concetto di ereditarietà sia anche una delle chiavi del successo della programmazione ad oggetti è sufficientemente chiaro. Data una classe, volendo effettuare una modifica all'intero progetto, è sufficiente modificare la classe base perché tutti gli oggetti che sono sua istanza cambino il proprio comportamento all'interno dell'intero progetto, senza per questo dovere copiare il codice in ogni parte del programma dove un certo oggetto viene utilizzato. Inoltre è possibile riutilizzare le stesse classi, estendole per scopi sempre più specializzati in altre soluzioni, senza dovere riscrivere d'accapo l'intero codice. Va da se che anche in ASP .NET la possibilità di scrivere dei controlli specializzato derivati da una classe base è fondamentale.
Prendiamo il caso di una funzione tutto sommato molto diffusa nelle applicazioni web: il pager.
Un controllo del genere consente di "paginare" un un elenco, dividendolo in più pagine ciascuna delle quali conterrà un certo numero di elementi. Ad esempio 100 elementi potrebbero essere "paginati" in 10 pagine da 10 elementi l'una, la navigazione sarebbe consentita grazie a 10 link numerici che puntano alle rispettive pagine. Questo genere di control è molto diffuso e spesso viene implementato come codice a parte, in ogni pagina in cui debba essere utilizzato.
L'unico vantaggio, peraltro discutibile, di una scelta del genere è la velocità di implementazione. Lo svantaggio è che se, per caso, vogliamo andarne a cambiare il funzionamento, per mostrare ad esempio solo 5 pagi-

ne di 20 elementi l'una ed evitare una lista lunghissima di link, ci tocca intervenire su tutte le pagine. Utilizzando ereditarietà e programmazione ad oggetti tenteremo di usare un solo controllo condiviso su tutte le pagine di modo che volendo modificarne il comportamento dovremo semplicemente effettuare la modifica su quell'unico controllo.

IL NOSTRO PRIMO CUSTOM CONTROL

Per il primo esempio ci limiteremo a creare un controllo custom che una volta inserito sulla pagina mostrerà l'orario e la data di Sistema. Questo controllo sarà derivato dalla classe base *Control* che, come già detto, contiene un metodo render che è anche il responsabile dell'output. Pertanto per far sì che il controllo stampi a video le informazioni desiderate, non dovremo fare altro che riscrivere il metodo *Render* della classe base. Questo tipo di operazione si chiama *"Override"*. Quello che faremo sarà cioè "sovrascrivere" il metodo render della classe base in una nostra classe da essa derivata per renderlo conforme alle nostre esigenze. Il codice è il seguente:

```
using System;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.ComponentModel;
namespace ASPItalia.com.UI.WebControls
{
  public class DateTimeControl : Control
  {
    protected override void Render(HtmlTextWriter
                                          output)
    {
      // scrivo sull'output
      output.Write("Data ed ora corrente: ");
      output.Write(DateTime.Now.ToString());
      // scrivo a video
      base.Render(output);
    }
  }
}
```

REGISTRARE ED UTILIZZARE IL CONTROL SULLA PAGINA

Realizzato, compilato e spostato nella directory */bin/* dell'applicazione, perché il nostro control sia utilizzabile, deve essere per prima cosa registrato sulla pagina. La sintassi è simile a quanto abbiamo visto, in alcuni numeri passati di questa rivista, a riguardo degli user control. In modo particolare, in questo caso la direttiva diventa:

```
<%@ Register TagPrefix="aspitalia" Namespace="
                    ASPItalia.com.UI.WebControls "
                  Assembly="aspitalia.controls"%>
```

L'attributo *TagPrefix* indica il prefisso da utilizzare per la dichiarazione nella pagina, mentre *Namespace* è ovviamente il riferimento al namespace che contiene i controls, dove *Assembly* indica invece il file risultante dalla compilazione.
Per poter utilizzare il nostro *DateTimeControl* all'interno della pagina dovremo inserire il seguente tag:

```
<aspitalia:DateTimeControl runat="server" />
```

Quello che per gli *user control* è il suffisso ed è liberamente specificabile, per i *custom controls* è invece rappresentato dal nome della classe che contiene il control stesso, quindi va scelto con cura in modo che sia primo di tutto esplicativo, ma che non sia troppo lungo né complesso da ricordare.

IL LINK ALLA SINGOLA PAGINA

Per il nostro control si è scelta la strada del codice HTML integrato all'interno dello stesso, anziché passare attraverso i control di ASP.NET. Questa scelta garantisce una migliore precisione nell'output, perché il meccanismo noto come adaptive rendering, che invia output diverso in base al browser, non entra in funzione.

```
// costruisco il singolo link
private string singlePage(int page, string text)
{
    if (text == null) text = page.ToString();
    return (CurrentPage == page)?
String.Concat("<b>[", page.ToString(), "]</b> "):
String.Concat("<a href=\"", String.Format(Url, page.ToString()), "\">", text, "</a> ");
}
```

UN CONTROLLO PIÙ COMPLESSO: IL PAGER

Vediamo come implementare un esempio per quanto riguarda il *"Pager"*, di cui abbiamo discusso ad inizio articolo. Il pager si presta facilmente ad essere "customizzato" con la tecnica della derivazione, perché si tratta di uno strumento in cui le varianti possono essere tantissime.
Il nostro esempio ha 3 proprietà pubbliche, una per il numero totale di pagine, una per la

I TUOI APPUNTI

pagina corrente ed un'ultima contenente la stringa che rappresenta l'URL, sul cui formato spenderemo tra un attimo due parole.
Ovviamente essendo un semplice control da agganciare ad altri (come un repeater), la logica di estrazione e di calcolo delle pagine totali va effettuata da un'altra parte.
Procediamo quindi con il creare il control. L'idea è sempre la stessa, deriveremo da control, creeremo dei metodi per far variare l'output in modo dinamico a seconda del numero di pagina e sovrascriveremo infine il metodo render per ottenere un output conforme alle nostre esigenze.
La funzione accetta due parametri, uno che rappresenta la pagina di cui deve essere creato il link ed un ulteriore che, se fornito, consente di associarvi un testo. Quest'ultimo ha senso di esistere sia per l'espandibilità futuro del controllo, ma anche e soprattutto per consentirci di aggiungere i due canonici link *"Avanti"* ed *"Indietro"*. All'interno c'è il controllo sulla pagina corrente, in modo che nel caso ci trovassimo su quest'ultima non venga generato il link, ma un testo in grassetto, che confermi all'utente dove si trova.
Ovviamente il funzionamento di questa funzione è parziale ed è necessaria aggiungerne una che consenta di verificare che quanto specificato nelle proprietà *PageCount* e *CurrentPage,* che rappresentano rispettivamente il numero di pagine e la pagina corrente, venga sfruttato al meglio per costruire l'elenco di link.

PAGER CON LE LETTERE DELL'ALFABETO

Creando una classe che derivi da quella che contiene il pager che abbiamo appena creato, possiamo costruire un nuovo pager che contenga i link alle lettere dell'alfabeto semplicemente sovrascrivendo la funzione buildPager, in modo che crei i link mettendo le lettere come descrizione, in questo modo:

```
// definisco le lettere in
                    un array
string[] letters = {"A",
    "B", "C", "D", "E", "F",
    "G", "H", "I", "L", "M",
    "N", "O", "P", "Q", "R",
    "S", "T", "U", "V", "Z"};

// faccio il ciclo
for (int i=0; i
        <letters.Length; i++)
lb.Append(singlePage(
            i, letters[i]));
```

Il risultato è che con pochissima fatica avremo riutilizzato buona parte del nostro codice per arrivare ad un risultato certamente diverso da quello per cui avevamo pensato il controllo padre.

Per prima cosa è dunque necessario costruire il link che consente di andare indietro, opzione possibile solo quando ci troviamo su una pagina che non sia la prima. Successivamente attraverso un ciclo aggiungeremo le altre pagine e per finire anche il link a quella successiva, caso possibile solo se non ci troviamo sull'ultima dell'elenco. Il tutto tradotto in codice:

```
namespace ASPItalia.com.UI.WebControls
{
  public class Pager: Control
  {
    // quante pagine
    private int pageCount;
    public int PageCount
    {
      get {return pageCount;}
      set {pageCount = value;}
    }
    // pagina corrente
    private int currentPage;
    public int CurrentPage
    {
      get {
        return currentPage;
      }
      set {
        currentPage = value;
      }
    }
    // url per paginazione
    private string url;
    public string Url
    {
      get {return url;}
      set {url = value;}
    }
    protected override void Render(HtmlTextWriter
                                        output)
    {
      output.Write(buildPager());
      base.Render(output);
    }
    // costruisco la sequenza di pagine
    private string buildPager()
    {
      // non ci sono altre pagine
      if (PageCount == 0)
      return String.Empty;
      // costruisco i link
      StringBuilder lb = new StringBuilder();
      // prossima pagina
      if (CurrentPage>1)
       lb.Append(singlePage(CurrentPage-1,
                        "<b>&lt;&lt;</b>"));
      // altre pagine
      for (int i = 1; i<=PageCount; i++)
        lb.Append(singlePage(i, null));
        // pagina successiva
      if (CurrentPage < PageCount)
        lb.Append(singlePage(CurrentPage+1,
                        "<b>&gt;&gt;</b>"));
      return lb.ToString();
    }
```

```
    // costruisco il singolo link
    private string singlePage(int page, string text)
    {
      if (text == null) text = page.ToString();
      return (CurrentPage == page)?
            String.Concat("<b>[", page.ToString(),
                                          "]</b> "):
      String.Concat("<a href=\"",
               String.Format(Url, page.ToString()),
                              "\">", text, "</a> ");
     }
   }
}
```

Il risultato è visibile nella figura ed essendo completamente generico, può essere riadattato per qualsiasi progetto e controllo che estragga i dati. Ed è proprio questo il bello dei *custom controls*.

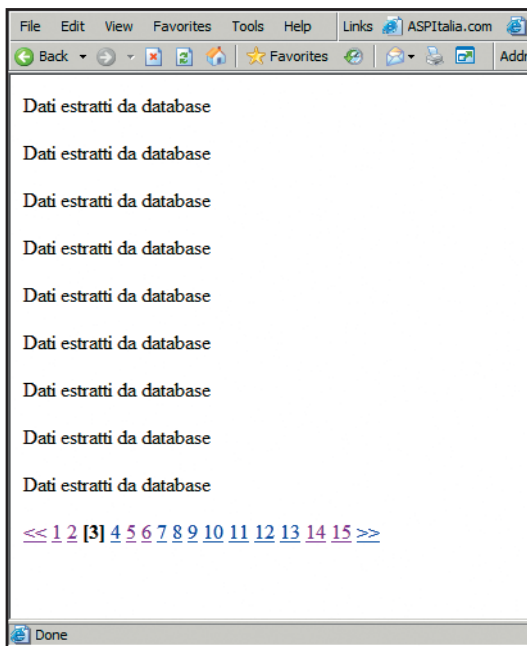



Fig. 1: *il nostro pager personalizzato all'opera*

Quanto all'utilizzo, vale la pena dare un'occhiata da vicino a questo pezzo della pagina, fornita come esempio insieme a tutti gli altri control creati in questo articolo:

```
<p><aspitalia:Pager id="Pager" url="pager.aspx?
         ricerca=test&page={0}" runat="server" />
<SCRIPT RUNAT="SERVER" LANGUAGE="C#">
void Page_Load()
{
  int page = 1;
  if (Request["page"] != null)
  page = Convert.ToInt32(Request["page"]);

  Pager.PageCount = 15;
  Pager.CurrentPage = page;
}
</SCRIPT>
```

Come si può notare l'URL contiene la stringa *"{0}"* al posto del numero effettivo di pagina. Se date uno sguardo alla funzione *singlePage*, noterete che in effetti si fa uso del metodo *Format* della classe *String*.
L'effetto di questo metodo è di cercare i parametri numerati, riconoscibili dalla parentesi graffe, e sostituirli con quanto specificato.
Nel nostro caso l'effetto è quello di rendere possibile la personalizzazione completa dell'URL di destinazione.

CUSTOM CONTROL O USER CONTROL?

Per ASP.NET, la scelta è semplice e tutto sommato si traduce in questo breve schema.
Se avete bisogno di un controllo che abbia quasi solo funzionalità visive, allora uno user control è perfetto, perché di fatto è un file sorgente con HTML mischiato ad un po' di codice, quando necessario.
Se invece volete un "vero" control, con possibilità di creare una classe, gestirla come tale, poterla ereditare, creandone di nuove, o più semplicemente avete bisogno di modificare un controllo esistente, come il repeater, perché vi offra funzionalità aggiuntive, allora la scelta cade sui Custom Control.
Inutile dirvi che, in fin dei conti, la vera differenza è che questi ultimi sono compilati in assembly, mentre gli User Control no. A voi la scelta.
ASP.NET consente due modalità principali per riutilizzare il codice

Custom control: sono classi vere e proprie, che ereditano da *Control* (o da un altro controllo di ASP.NET) e sono principalmente composte da codice C# (o VB.NET);

User control: sono pezzi di pagina, composti più da HTML che da codice vero e proprio.

I vantaggi dell'uno o dell'altro approccio sono evidenti e consistono principalmente nell'essere utili in scenari diversi.

CONCLUSIONI

Questa è proprio una delle aree in cui ASP .NET vi mette un solo ed unico limite: la vostra fantasia. O se volete, le vostre necessità di costruire applicazioni potenti, complete e soprattutto flessibili e facilmente espandibili.

Daniele Bochicchio

L'AUTORE

**Daniele Bochicchio è il content manager di ASPItalia.com, community che si occupa di ASP.NET, Classic ASP e Windows Server System. Il suo lavoro è principalmente di consulenza e formazione, specie su ASP.NET, e scrive per diverse riviste e siti.
È Microsoft ASP.NET MVP, un riconoscimento per il suo impegno a supporto delle community e per l'esperienza maturata negli anni. Il suo blog è all'indirizzo** http://blogs.aspitalia.com/daniele

SOFTWARE SUL CD

APACHE ANT 1.6.5

Come Make ma basato su Java

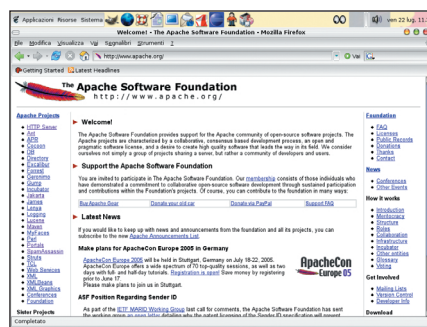
I programmatori C++ sono abituati ad usare l'utility "Make" per la compilazione dei propri programmi. Come molti sapranno Make come le altre utility dello stesso genere, ad esempio nmake e jam vengono utilizzate in fase di compilazione per ricostruire le dipendenze dei vari file e classi che compongono un progetto. Ant svolge un compito simile ma aggira le limitazioni imposte dagli altri programmi. Piuttosto che analizzare file di testo che contengono le regole di dipendenza analizza classi java e file xml per ricostruire il percorso di compilazione. Si tratta di un tool completamente scritto in java che viene ormai usato in più di un progetto.

Directory :/Ant

APACHE 1.3.33/2.0.54

Il Web Server che fa girare Internet

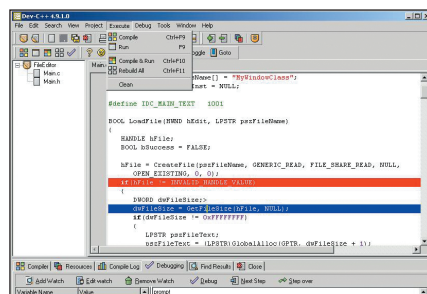
Apache è stato uno dei primi Web Server a fare la sua comparsa su Internet e nel tempo ha conquistato una posizione piuttosto rilevante sulla rete. Al momento la sua diffusione è tale che si può dire che una larga parte del world wide web gira sul codice di Apache.

Si tratta di un Web Server leggero e affidabile, molto testato, ed estendibile per mezzo di moduli esterni. Gira sia in ambiente Windows che in ambiente Linux. Nel primo caso subisce la concorrenza di IIS rispetto al quale manca in parte per supporto a .NET e .ASP. Nonostante questo Apache risulta usato anche negli ambienti Microsoft in quanto si rivela uno dei tool essenziali e più facili da configurare per il supporto a prodotti OpenSource.

Sicuramente si tratta di un prodotto da usare e altrettanto sicuramente vi troverete a usarlo in compagnia di PHP, MySQL oppure PostgreSQL come base portante delle vostre Web Application.

Directory: /Apache

DEV C++

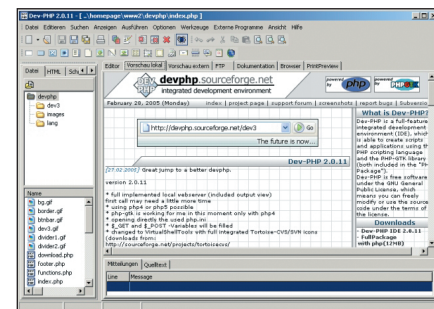
Il più amato dai programmatori C++

C++ è ancora oggi uno dei linguaggi più diffuso per lo sviluppo di software multipiattaforma. Se da un lato nel tempo ha subito l'attacco di linguaggi più semplici da imparare e da utilizzare, d'altro canto rimane l'unica soluzione percorribile quando volete sviluppare applicazioni che dialoghino a basso livello con le periferiche, o quando volete ottenere un compilato veloce e affidabile.

DevC++ è un IDE di programmazione per C++ completamente freeware che si pone come alternativa economica ai vari compilatori commerciali in ambiente Windows. Di default supporta minggw ma può anche essere utilizzato con altri compilatori C++. Si tratta di un ambiente completo, leggero e funzionale che per le sue caratteristiche è diventato nel tempo un punto di riferimento per la community degli sviluppatori C++.

Directory :/DevCPP

DEV PHP

L'IDE gratuito per PHP

PHP è uno dei linguaggi principe per Internet. Le caratteristiche che l'hanno reso così diffuso sono, una bassa curva di apprendimento, la completezza delle funzioni esposte, l'integrazione con ogni tipo di database e per entrare più nel dettaglio, con la versione 5, anche classi, ereditarietà, polimorfismo e le altre caratteristiche tipiche dei linguaggi ad oggetti evoluti.

Si può programmare in PHP anche utilizzando il notepad, ma certamente un buon ambiente di sviluppo favorisce la produttività. In questo senso DevPHP rappresenta un'ottima soluzione. Si tratta di un IDE evoluto, completamente gratuito, dotato di tutte le funzionalità classiche che rendono semplificato lo sviluppo del

codice, dalla code complexion alla sintax hilighting.
Directory :/DevPHP

DRUPAL 4.6.2

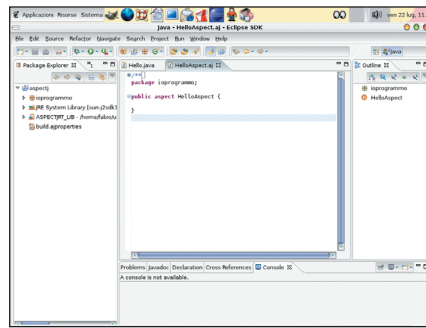
Il principe dei blog

Drupal è una piattaforma per la costruzione di Blog. Ogni utente registrato al sistema diventa automaticamente proprietario di un blog, può postare, cancellare, impostare l'aspetto del proprio spazio.
Il successo di Drupal è dovuto probabilmente alla sua modularità, infatti è possibile espandere le funzioni base includendo in modo quasi automatico dei moduli esterni.
D'altra parte sviluppare un modulo per Drupal non è un'operazione complessa.

È sufficiente avere delle buone basi di PHP ed addentrarsi nel framework di sviluppo del sistema che comunque risulta ben strutturato e facile da apprendere. Per tutti questi motivi Drupal ha conquistato nel tempo una posizione di assoluto rilievo nel panorama dei sistemi dedicati ai blog, e di blog ormai ne esistono davvero tantissimi.
Directory: /Drupal

ECLIPSE 3.1

L'ambiente tuttofare

Eclipse è un IDE "Aperto" multipiattaforma e completamente scritto in Java. Per aperto si intende un sistema che può essere facilmente espandibile per mezzo di moduli. Perciò se da un lato inizialmente si presenta pronto per favorire lo sviluppo di applicazioni Java, dall'altro lato tramite moduli può diventare un comodo IDE per PHP oppure per C++ oppure per XML, oppure per qualunque altro linguaggio che sia supportato da un modulo.

In questo numero di ioProgrammo lo abbiamo utilizzato per presentare AspectJ, una tecnica che consente di fare "dialogare" in modo particolarmente efficace le applicazioni Java
Directory :/Eclipse

IRRLICHT 0.11.0

Il motore 3D per lo sviluppo di VideoGames

Da quando abbiamo cominciato a parlare di Irrlicht sulle pagine di ioProgrammo è stata un escalation di diffusione. Il merito è sicuramente la potenza e la facilità d'uso di questo straordinario Engine 3D. In questo numero di ioProgrammo ne abbiamo parlato in relazione all'applicazione di effetti di parallax mapping su sequenze animate.

In condizioni normali i calcoli per l'applicazione del parrallax mapping sarebbero piuttosto complessi, è qui che si vede la potenza di Irrlicht che mette a disposizioni poche semplici primitive per la realizzazione di questo effetto. Chiaramente questo è solo un esempio della ricchezza del motore, irrlicht ha davvero tutti i numeri per diventare un punto di riferimento in questo settore.
Directory :/irrlicht

JDK 1.5.0.04

Il compilatore Java

Non ci sono molte parole da spendere per il JDK di Sun. Si tratta del compilatore Java. Al di la di poche iniziative OpenSource e di un tentativo di Microsoft non esistono alternative a questo strumento se volete programmare in Java. Quello che vi presentiamo è il quarto upgrade alla versione 5 del compilatore, recentemente rilasciato e che contiene alcuni interessanti Bug Fix. Non una rivoluzione ma sicuramente un passo ulteriore in avanti per questo compilatore che ormai detiene quote di mercato così elevate, tali da renderlo un must per chiunque si accinga a sviluppare un qualche progetto specialmente se multipiattaforma
Directory:/ Java

LUCENE 1.4.3

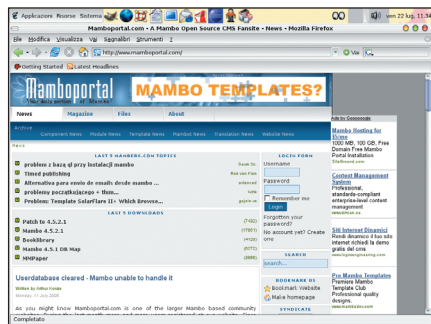
Il segugio cercatutto

Prima Google ha rilasciato la sua Desktop Search Bar che consentiva di indicizzare il contenuto del vostro hard disk per potere effettuare delle ricerche veloci. Un po' come il famoso motore di ricerca, ma locale al vostro PC. Subito dopo si sono scatenati, ovviamente i concorrenti OpenSource. Lucene è uno di questi. Sulla sua base è nato ad esempio il progetto Beagle, uno strumento scritto in Mono che sfrutta lucene per creare un proprio motore di ricerca. Tocca a voi adesso migliorare beagle, oppure utilizzare lucene all'interno dei vostri software. Noi aspettiamo solo che ci mandiate le vostre soluzioni. La tecnica è davvero interessante e stimola la fantasia per quanto riguarda i tool di sviluppo.
Directory :/Lucene

MAMBO 4.5.2.3

Il re dei Content Management Systems

A fare da apripista fu PHPNuke, seguirono poi i vari Xoops e poi centinaia di fork sulla base dei quali è nato il folto regno dei CMS. Nell'ultimo anno il trono di leader è stato conquistato da Mambo, un CMS scritto in PHP che per le sue caratteristiche si è subito imposto sui concorrenti. In particolare mambo si dimostra piuttosto flessibile per la generazione dei temi, ma anche per la modularità con cui è stato progettato, e per l'interfac-

cia di amministrazione tramite la quale è possibile compiere praticamente ogni operazione.

In sostanza si tratta di un ottimo progetto per uno sviluppatore che volesse fornire a qualche suo cliente un sistema di CMS altamente personalizzato, ma anche per l'utilizzatore che si troverebbe a che fare con un un'interfaccia semplice ma potente.
Directory:/ Mambo

MEDIAWIKI 1.4.7

Il creatore di enciclopedie

Da quando è stato rilasciato il primo Wiki, questo genere di sistema si è imposto in tutte quelle situazioni dove è necessaria generare documentazione. Per essere precisi un wiki è una web application dotata di un linguaggio interno che si autoreferenzia. Ad esempio per scrivere un nuovo contenuto è sufficiente puntare il browser verso una pagina inesistente. Il wiki intercetterà la richiesta e produrrà una form all'interno della quale potete inserire i vostri contenuti.

Allo stesso modo in un documento successivo la pagina appena prodotta potrà essere referenziata semplicemente inserendo uno specifico tag all'interno del nuovo documento.
Il sistema, nella sua semplicità, si è rivelato molto utile, tanto che sono nati diversi progetti ormai famosi, e tanto che quasi tutti i siti di supporto al software usano il sistema dei wiki. Il più famoso wiki in circolazione è probabilmente wikipedia: http://it-.wikipedia.org
Directory :/MediaWiki

MONO 1.1.8

Il rivale opensource di .NET

Mono è nato come progetto Open-Source per la piattaforma .NET. In realtà .NET non è una tecnologia esclusivamente proprietaria di Microsoft, quanto un progetto condiviso da più realtà. Se da un lato Microsoft lo ha adottato come principe per i suoi futuri ambienti di sviluppo e seriamente sta producendo strumenti veramente evoluti che la pongono in condizione di leadership in questo settore, d'altra parte esistono delle valide alternative come Mono.
Il vantaggio di mono è che è multi-piattaforma, potete sviluppare in C# sia per linux che per Windows. Lo svantaggio è che non potete utilizzare classi troppo legate al sistema operativo altrimenti perderete il vantaggio della portabilità. Si tratta comunque di un sistema da tenere in seria considerazione se desiderate sviluppare applicazioni portabili.
Directory :/mono

MYSQL 4.1.13/5.0.9

Il database del Web

MySQl è nato molti anni fa con l'intento di esportare poche ma utili funzioni. Il prodotto doveva essere semplice e veloce. Nel tempo queste due caratteristiche sono rimaste costanti ma il numero di funzioni che compongono MySQL si è arricchito in modo straordinario. Fino a qualche tempo fa si poteva imputare a MySQL l'assenza di qualche caratteristiche che si è rivelata molto utile per altri database, come ad esempio il supporto alle transazioni, ai trigger, alle stored procedure, alle view. Molti di questi limiti sono stati aggirati con la versione 4, molti altri verranno invece colmati nella versione 5. Nonostante l'assenza di queste caratteristiche in ogni caso MySQL è diventato il leader nel suo campo, cioè le web application, nel corso degli anni. Segno evidente che il prodotto era ben tarato sulle esigenze dei suoi utilizzatori, l'aggiunta delle nuove caratteristiche che ne completano e le estendono le funzionalità non può che rappresentare un ulteriore passo in avanti per questo straordinario db.
Directory:/ mysql

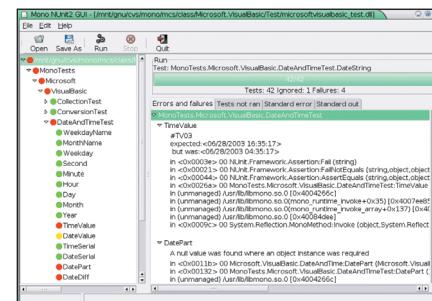
NASM 0.98

Il compilatore Assembly

Si d'accordo nessuno programma più in assembly, si altrettanto d'accordo la produttività è rappresentata dai RAD, ambienti superevoluti che mascherano all'utente la complessità dell'assembly. Tutto questo si paga in prestazioni e in controllo. Dunque se volete ottenere il meglio da una routine o se volete pilotare a basso livello una periferica avete una sola via d'uscita: ASSEMBLY. Il Nasm è uno dei pochi compilatori ancora disponibile per questo linguaggio. Se volete avere potere e controllo dovete usarlo!
Directory: Nasm

NUNIT 2.2.0

La suite per il test dei programmi

Ce ne parla approfonditamente su questo stesso numero di ioProgrammo Gianluca Negrelli nel bell'articolo sull'Extrme Programming.

Nunit consente di elaborare test precisi a cui sottoporre il software per l'analisi delle funzionalità. Con questo tipo di testi si possono riuscire ad abbattere drasticamente i tempi per l'assistenza e il mantenimento che spesso rappresentano uno dei fattori di maggior spesa nell'economia della produzione di un'applicazione. Se ben usato Nunit può rappresentare un'ottima soluzione che garantisce un drastico abbattimento di questi costi.
Directory: /Nunit

PHP 4.4.0/5.0.4

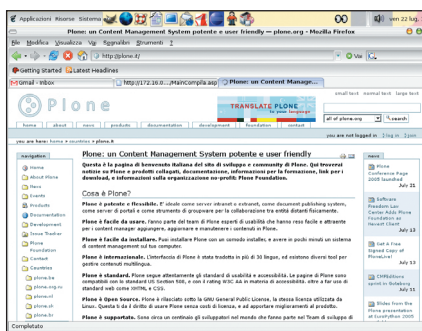
Il linguaggio del Web

PHP è probabilmente uno dei più po-

polari linguaggi per lo sviluppo di applicazioni Web. Un numero veramente ampio di web application specialmente OpenSource è basato su questo linguaggio.
Fino a qualche tempo fa soffriva di un supporto agli oggetti non completo che lo rendeva meno efficace nello sviluppo di applicazioni di grandi dimensioni, anche se questo limite non gli ha impedito di diventare un leader nel suo settore. La versione 5 ha completamente rivoluzionato il modo di intendere la OOP in PHP aggiungendo un supporto alla programmazione Object Oriented sufficientemente avanzato.
Directory: /PHP

PLONE 2.0.5

Il Cms professionale

Plone nasce come progetto basato sull'application server Zope. Come tale si pone come uno strumento piuttosto professionale che fa della sicurezza e dell'affidabilità il suo focus. Non per questo esporta poche funzioni, anzi, al contrario Plone per numero e qualità di funzioni non è secondo a nessuno.

È un CMS ma anche un wiki, un forum, una piattaforma per blog, un sistema completo da utilizzare in ambienti dove è importante potere contare su un sistema altamente affidabile prima che facilmente estendibile
Directory: /plone

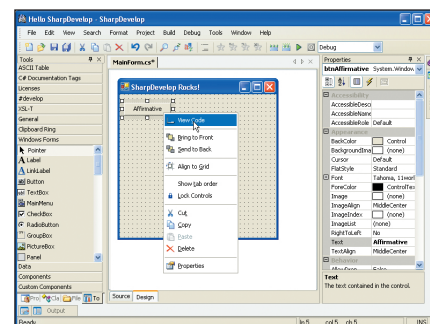
POSTGRESQL

Completo e OpenSource

Fra i database OpenSource non ne abbiamo trovato ancora uno in grado di competere per quantità di funzioni implementate! e a dire il vero pochi sistemi commerciali reggono il confronto con PostgreSQL.
Al di là della velocità e dell'affidabilità del prodotto sono le sue caratteristiche ad averci impressionato. Foreign Keys, stored procedure, view, ereditarietà delle tabelle sono solo alcune delle caratteristiche che fanno di PostgreSQL un database completo. Particolarmente interessante è l'estendibilità del prodotto a mezzo di moduli esterni. Ne esistono già tantissimi che implementano praticamente tutto lo scibile sui database. Le caratteristiche interne al db più quelle aggiunte da moduli esterni ne fanno un prodotto completo e fuori dal comune.
Directory: /PostgreSQL

SHARPDEVELOP 1.1.0

L'ambiente gratuito per programmare in .NET

SharpDevelop ha fino ad ora rappresentato l'unica via d'uscita per chi voleva sviluppare per il sistema Windows utilizzando .NET ma non i costosi compilatori di Microsoft.

Si tratta di uno strumento che con il tempo si è evoluto fino a raggiungere dignità di ambiente RAD e la completezza che si deve a questo tipo di ambienti.
Con SharpDevelop è possibile programmare in C#, è stato fatto qualche passo in avanti verso Visual Basic .NET tuttavia il supporto non è ancora così completo come per il primo caso. Si tratta di un IDE da utilizzare se volete programmare per la piattaforma .NET ma non avete bisogno delle complessità di Visual Studio
Directory /sharpdevelop

SNORT 2.3.3

Il sistema che rileva gli intrusi

Snort è un Intrusion Detection System, per gli amici IDS. Ovvero un sistema che si mette in ascolto sulla vostra scheda di rete e rileva tutti i pacchetti che vi passano attraverso. Ogni pacchetto viene confrontato con una serie di regole. E se qualche pacchetto viene identificato come potenzialmente pericoloso viene generato un Alert che segnala all'amministratore che c'è la possibilità che un qualche attacco sia in corso verso il proprio sistema.
Un vero 007 che non lascia scampo a chi cerca di utilizzare in modo improprio la potenza di TCP/IP ma anche di sfruttare le enormi falle che ogni giorno vengono scovate all'interno dei protocolli e dei software più comuni
Directory :/snort

TOMCAT 5.5.9

L'application server per le JSP

Tomcat è un Web Server, ma soprattutto è un application server che consente di fare girare sul web applicazioni Java scritte come JSP o come Servlet.
Si caratterizza come un sistema piuttosto complesso ma anche sicuro ed affidabile, d'altra parte rappresenta uno dei pochi modi per implementare web application basate sulla potente tecnologia del linguaggio di Sun
Directory /Tomcat

ZOPE 2.8.0

L'application Server per Python

Zope, fra gli application server occupa sicuramente una posizione difficile. Si tratta di un prodotto complesso, potente, estremamente flessibile, che per numero di funzioni rivaleggia con i concorrenti più conosciuti. Viene utilizzato in un innumerevole serie di prodotti professionali, tuttavia si fonda sul linguaggio python che sicuramente sul web non occupa ancora una posizione di rilievo nonostante le sue caratteristiche siano impressionanti. Le statistiche dicono che in America Python è il linguaggio che è maggiormente cresciuto in quanto a diffusione nell'ultimo anno.
Se è vero che gli USA dal punto di vista dello sviluppo software fanno da apripista non potremo non fare i conti con Python da qui a breve tempo
Directory: /Zope

Programmare i Monitor

Non è di video che stiamo parlando, ma di una tecnica usata dal kernel dei sistemi per impedire lo "stallo": quando due processi tentano di accedere contemporaneamente a una risorsa condivisa

Lo scopo di questo articolo sarà individuare dei metodi per far sì che due processi possano condividere una risorsa in modo efficiente. Al solito partiremo dall'esempio del processo A che tenta di stampare contemporaneamente al processo B. Ovviamente la stampante non può stampare contemporaneamente i dati provenienti da due processi diversi. Perciò le richieste "concorrenti" vanno gestite.

Avevamo già individuato un metodo nei numeri passati di ioProgrammo, ovvero quello delle regioni critiche. Avevamo definito una regione critica come una porzione di processo che potenzialmente potrebbe accedere a risorse condivise, ed avevamo poi individuato alcune regole ad esempio due processi non possono trovarsi contemporaneamente nella loro regione critica.

Avevamo infine individuato algoritmi basati su semafori per gestire queste regole. Naturalmente l'uso delle regioni critiche prevede che i processi debbano parlare fra di loro e comunicarsi a vicenda l'entrata o l'uscita da una regione critica.

Un semaforo è in sostanza una variabile che gestisce la sincronizzazione fra processi. Prima di entrare in una regione critica un processo controlla il semaforo, se lo stato è tale che si è certi che nessun altro processo è in una regione critica allora consente l'accesso alla risorsa e varia lo stato del semaforo affinché nessun altro processo possa accedere, quando esce dalla regione critica riporta lo stato del semaforo a un valore tale che gli altri processi possano accedere.

È tutto molto bello, ma le regioni critiche e i semafori fanno sì che la "sincronizzazione" fra processi debba essere programmata manualmente. Una soluzione più sofisticata è quella dei "Monitor".

DALLE REGIONI CRITICHE AL MONITOR

Il monitor come la regione critica è uno strumento per l'attuazione della concorrenza tra processi ed in particolare per la condivisione di tipi di dati astratti tra essi. La sintassi di un monitor, come vedremo, è semplice. Si tratta di definire un tipo di dato astratto monitor. Ad esempio la classe che individua una stampante potrebbe essere definita come un monitor. A questo punto il monitor verrebbe condiviso da più processi. La peculiarità di questo nuovo strumento è che un solo processo per volta può essere attivo dentro un monitor. Come detto, il monitor a differenza delle regioni critiche, si occupa in automatico della gestione della sincronizzazione. A tale scopo è stato introdotto un nuovo costrutto, indichiamolo con il suo nome originale anglosassone: condition.

Ecco come si può dichiarare una variabile condition.

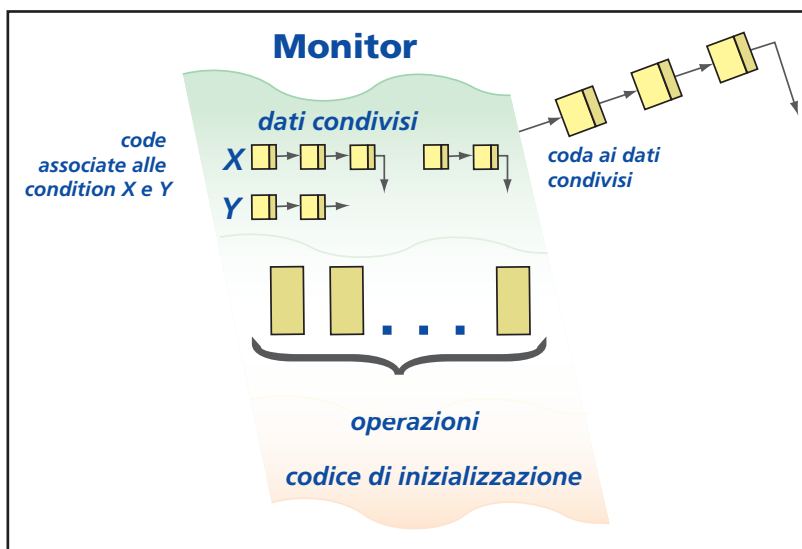



Fig. 1: **Schematizzazione dei monitor con variabili condition**

```
var x, y : condition;
```

Per variabili così definite si possono richiamare due sole fondamentali funzioni: wait e signal. In altri termini le variabili condition possono essere viste come tipi di dati astratti a cui sono associate due operazioni.

```
x.wait;
x.signal;
```

Quando un processo invoca la prima delle funzioni (wait) esso viene sospeso fin quando un altro processo non lancia la seconda (signal). La signal ha quindi il compito di riattivare i processi in precedenza sospesi. Se tale funzione è richiamata su processi non sospesi allora non produrrà alcun effetto. Come si può notare, hanno una fortissima analogia con le operazioni associate ai semafori, la P e V.
Tuttavia una differenza sostanziale tra signal e V è immediatamente rilevabile, infatti, la V dei semafori ha sempre effetto.

IL BIVIO

Supponiamo che un processo *P* lanci una *x.signal* e che preventivamente un altro processo *Q* sia stato sospeso mediante *x.wait*. La signal avrà l'effetto di riattivare il processo *Q*, qualora sia possibile, in tal caso *P* dovrà attendere. Tuttavia la scelta di attivare o meno *Q* e sospendere *P* dipende da una serie di circostanze per cui potremmo anche adottare una delle due soluzioni:

a) *P* attende finché *Q* libera il monitor, o attende per altre condition.

b) *Q* attende finché *P* libera il monitor, o attende per altre condition.

Giacché *P* già impegna il monitor sarebbe ragionevole la soluzione b. In tal caso però potrebbe risultare oltremodo lunga l'attesa del processo Q.
La soluzione adottata da Hoare è la prima. Brinc Hansen ha invece adottato un compromesso. Quando il processo P segue la signal immediatamente libera il monitor, e Q è immediatamente riattivato. Questo schema è meno potente della soluzione di Hoare poiché un processo non può attivare più signal durante una singola chiamata. Chiariamo il concetto sviluppando un monitor che simula un semaforo binario.

```
type semaphore = monitor;
var occupato: boolean;
            z: condition;
procedure P;
begin
            if occupato then z.wait
            occupato:=true;
end;
procedure V;
begin
            occupato:=false;
```

SEMAFORI IN PILLOLE

Un semaforo è una variabile S che eccetto la fase di inizializzazione è manipolata da due sole operazioni atomiche: P e V. Esaminiamo la definizione base delle due funzioni.

```
P(S): while (S<=0) do;
         S:=S+1;
V(S):    S:=S+1
```

Come visto nell'articolo specifico (ioProgrammo 93) esistono implementazioni più efficienti. Tale codice serve a capire come si manipola la variabile S e cosa si intende per P e V. In estrema sintesi P produce un attesa e V sblocca, il tutto in funzione del valore della variabile S. Osserviamo adesso come questo costrutto sia usato nella gestione delle sezioni critiche. Poi esamineremo nei particolari l'implementazione. Si suppone che n processi condividano un semaforo di nome mutex, che ha valore iniziale pari a 1. Ogni processo gestisce la sezione critica attuando il ciclo infinito proposto di seguito:

```
repeat
  P(mutex)
  <sezione critica>
  V(mutex)
  <codice rimanente>
until false
```

La sezione critica è protetta dalla presenza di un semaforo che impedisce al processo di entrare se mutex è occupato. Una volta terminato lo sfruttamento della sezione la si libera con V. Esaminiamo un altro esempio in cui si abbiano due processi P1 e P2 che eseguono rispettivamente le due istruzioni S1 e S2. Supponiamo di avere come vincolo che l'esecuzione di S2 debba avvenire soltanto dopo che S1 sia completamente terminata. Il processo può essere sincronizzato mediante l'uso di un semaforo. Sia sincr il semaforo citato con valore iniziale 0, la soluzione si otterrà facilmente modificando di poco i due processi P1 e P2.

```
(* P1 *)
S1;
V(sincr);
(* P2 *)
P(sincr);
S2;
```

Poiché sincr è inizializzato a 0, P2 eseguirà il suo statement S2 soltanto dopo che verrà superata la funzione P, ossia quando P1 invocherà V(sincr), quindi dopo S1.

```
        z.signal;
end;
(*inizializzazione*)
begin
        occupato:=false;
end.
```

La variabile occupata indica lo stato del semaforo. Quando si richiama P il semaforo passa allo stato di occupato, la variabile omonima sarà vera. Se vengono tentate altre operazioni P i processi che le hanno lanciate devono attendere fin quando non si esegue una V. Da notare che V porta occupato sempre a falso. Se c'è un processo in attesa questo resetta occupato immediatamente a vero, altrimenti rimane falso.

IL RITORNO DEI FILOSOFI

Un altro interessante problema la cui soluzione si può ricercare dall'attuazione dei monitor è quello dei filosofi affamati. Affronteremo il caso più complesso dei filosofi affamati, per il quale la soluzione non produce mai deadlock (stallo). Quindi il filosofo prenderà la bacchetta solo nel caso in cui siano disponibili entrambe le bacchette vicine. Per il resto si procede nella maniera solita. Ogni filosofo può trovarsi in uno tra tre stati: pensa, ha fame, mangia. La struttura dati associata è quindi la seguente:

```
var stato:  array [0..4] of (pensa, ha_fame, mangia);
```

si tratta di un array di enumerati. Il generico filosofo i potrà passare nello stato di mangia (in italiano corretto mangerà) solo se entrambi i filosofi a suo fianco non stanno mangiando. Tale condizione, come accennato in precedenza, previene il *deadlock*. Ovviamente, ciò porterà alla conseguenza che mai due filosofi vicini possano contemporaneamente mangiare. Sarà necessario anche prevedere un monitor e le variabili condition ad esse associate.

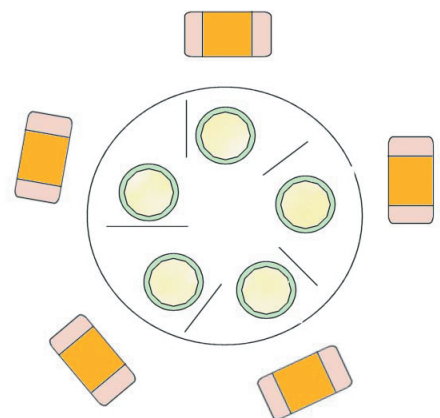
```
var fil: array [0..4] of condition;
```

Tali variabili permettono al filosofo i di attendere quando sono affamati, ovvero si trovano nello stato di *ho fame*. Le risorse, che nello specifico sono le bacchette dei filosofi orientali (provenienza che giustifica l'uso di bacchette per mangiare!), le controlleremo con monitor.

Ecco il codice nella sua completezza.

```
type filosofo_affamato = monitor
var stato:  array [0..4] of (pensa, ha_fame, mangia);
var fil: array [0..4] of condition;
procedure prendi (i: 0..4);
begin
        stato[i]:=ha_fame;
        test(i);
        if stato[i]<> mangia then fil[i].wait;
end;
procedure lascia (i: 0..4);
begin
        stato[i]:=pensa;
        test(i-1 mod 5);
        test(i+1 mod 5);
end;
procedure test (k: 0..4);
begin
        if (stato[k-1 mod 5] <> mangia) and
          (stato[k] = ha_fame) and
          (stato[k+1 mod 5] <> mangia)
        then begin
                stato[k]:=mangia;
                fil[k].signal;
            end;
end;
(* Inizializzazione *)
begin
        for i:=1 to 4 do
        stato[i]:=pensa;
```

FILOSOFI AFFAMATI

È un fondamentale esempio nel campo della programmazione concorrente. Per molti ne descrive addirittura il paradigma.
La soluzione in ambito concorrente è un indispensabile riferimento in una ampia casistica di problemi.
Descriviamo il problema.
Si tratta di un banchetto di *n* filosofi che svolgono le attività di pensare, mangiare ed essere affamati. Attingono ad un vassoio di riso e poiché sono cinesi usano le bacchette, due per la precisione. Anche le bacchette come i filosofi e i vassoi sono *n*, per cui un filosofo può mangiare solo se trova libera una bacchetta a destra e una sinistra.
È facile l'analogia con il mondo della programmazione. Le bacchette sono risorse e i filosofi processi che consumano risorse.
L'idea alla base della soluzione è associare un semaforo o una variabile condition dei monitor ad ogni bacchetta.
Si tratterà quindi di definire un array di *n* semafori o di condition.

La scena di riferimento per il problema dei filosofi affamati"

```
end.
(*Possibile uso di un'istanza di filosofo_affamato di
                                              posto i*)
 var fa : filosofo_affamato;
…
fa.prendi(i);
…
            <mangia>
…
fa.lascia(i)
```

La variabile *fa* è un'istanza di filosofo_affamato, ossia il monitor, array di filosofi, gestito in modo concorrente.
Quando un filosofo prende la bacchetta, si attiva lo stato di attesa su di essa, ponendo in wait l'iesimo filosofo. La risorsa è occupata e nessuno può usarla. Quando si mangia e si termina, si rilascia la risorsa con una signal. La procedura lascia, inoltre, porta nello stato di pensa il filosofo. La routine test viene invocata sia nella procedura prendi, per verificare se ci sono le condizioni per acquisire la risorsa, sia nella procedura lascia appunto per rilasciare le risorse.

ARRAY CIRCOLARI E OPERAZIONE MOD

Quando si ha a che fare con array circolari bisogna gestire una semplice quanto indispensabile situazione, ovvero che il successivo elemento dell'ultimo sia il primo. Per implementare ciò è sufficiente gestire un'unica variabile indice che può essere incrementata o decrementata a piacimento senza doversi preoccupare se valica i limiti della dimensione dell'array. Unica accortezza e fare riferimento all'indice con l'operazione i mod n. Dove i è l'effettivo indice soggetto a variazione n è la dimensione dell'array; mod è l'operazione di resto sulla divisione intera tra i e n. Così il risultato sarà sempre compreso nell'intervallo 1..n. Se ad esempio i vale n+1, ovvero l'indice ha oltrepassato di uno la dimensione dell'array, applicando la formula descritta si accederà all'elemento di posto 1, andando a realizzare appunto la circolarità della struttura.

IMPLEMENTAZIONE MEDIANTE SEMAFORI

Vediamo come i monitor si possano progettare e sviluppare utilizzando il meccanismo più a basso livello dei semafori. Per ogni monitor un semaforo mutex è inizializzato a 1. La primitiva P(mutex) sarà eseguita prima di entrare nel monitor, mentre la V(mutex) dopo, quando bisogna liberarlo. È necessario introdurre altre variabili per implementare al meglio il monitor. Un altro semaforo next, inizializzato a 0, viene usato per consentire a un processo che fa signal di sospendere se stesso. Questo perché se un processo che fa signal deve attendere finché il processo riattivato non sia libero o in attesa. Una variabile intera next_cont fornirà il numero di processi sospesi su next. Così una generica procedura M del monitor potrà essere sostituita dal seguente codice.

```
P(mutex);
…
corpo di M
…
if next_cont >0 then V(next)
else V(mutex);
```

Nel ramo then si entra quando ci siano più processi in coda. Nel ramo else se i processi sono finiti. Esaminiamo adesso come si implementano gli elementi chiave dei monitor, ovvero le variabili condition. Ogni variabile x di tipo condition è associata ad un semaforo x_sem e a un intero x_cont entrambi inizializzati a zero. Ecco come x.wait può essere implementata:

```
x_cont:=x_cont+1;
if next_cont>0
then V(next)
else V(mutex);
P(x_sem);
x_cont:=x_cont-1;
```

La prima è una fase preliminare. Sblocca un processo che aveva fatto signal. Subito dopo con la P su x_sem si attende, ed infine si decrementa il numero di processi in attesa su x. L'operazione di signal si implementa come segue:

```
if c_cont >0
then begin
            next_cont:=next_cont + 1;
            V(x_sem);
            P(next);
            next_cont:=next_cont – 1;
     end;
```

Da notare che questa implementazione di monitor è applicabile a entrambe le definizioni di monitor; sia quella di Hoare sia quella di Brinc Hansen.

WAIT CONDIZIONALI

Prima di esaminare un altro interessante problema risolvibile con in monitor, poniamoci una domanda. Supponiamo che più

processi siano sospesi su una variabile condition x e che una x.signal (all'interno di qualche processo) sia lanciata. Ci chiediamo: quale processo sarà il prossimo ad essere riattivato. A questo punto dovremmo rivisitare le tecniche di assegnazione della CPU, in effetti sono diverse. Ci limitiamo a ricordare il first come first served che segue una logica FIFO (first in first out) pura.
Per la quale il primo che ne fa richiesta sarà il primo ad essere accontentato nella sua necessità di utilizzo della risorsa. Può accadere così che un processo che ha una elevata richiesta, in termini di tempo, di una risorsa si trovi ad essere servito per primo, rallentando di fatto tutto il lavoro degli altri processi.
Le wait condizionali sono state introdotte sempre da Hoare, per rendere più flessibile il criterio di assegnazione delle risorse.
Esse si presentano nella forma:

```
x.wait(k);
```

Dove k è un intero che serve a valutare il criterio di scelta dei processi in coda alla risorsa. Quindi il numero k stabilisce la priorità; ovviamente ogni processo sospeso sulla conditon ne avrà uno.
Quando si esegue la x.signal tra i processi sospesi in coda si sceglie quello con k minore. Tale costrutto si presta naturalmente ad implementare un comune metodo di assegnazione di CPU e risorse in generale, il short job first. Tale metodo serve prima i processi che richiedono la risorsa per minor tempo, accodando le restanti. Senza entrare nel merito dei singoli metodi e della loro efficienza osserviamo come si può sviluppare l'ultimo descritto con il nuovo costrutto introdotto.

```
type short_job_first =monitor;
var occupato : boolean;
              x : condition;
procedure acquisisci  (tempo:integer);
begin
        if occupato then x.wait(tempo);
           occupato:=true;
end;
procedure rilascia;
begin
           occupato:=false;
           x.signal;
end;
(* inizializzazione *)
begin
           occupato:=false;
end.
```

Tra i processi sospesi si sceglie quello che ha il parametro di priorità della wait minore. Risulta evidente che il tempo di richiesta della risorsa sarà l'elemento su cui ordinare la coda sulla x in modo da servire i processi per così dire più veloci. Sfruttando appieno il costrutto di Hoare.
Nel rilasciare la risorsa basterà portare occupato a false e lanciare una nuova signal. Ovviamente, nel caso reale, a volte, il tempo di utilizzo della risorsa non si conosce a priori ma è il frutto di una stima. Ma per il problema che ci interessava risolvere, questo aspetto diventa un semplice dettaglio.

CONCLUSIONI

Si è concluso il percorso che ci ha condotto negli interessanti e a volte intricati sentieri della programmazione concorrente. La trattazione si è composta di una serie di passi che nel corso degli anni gli esperti di informatica hanno condotto per rendere i sistemi operativi multiprogrammati (multitasking) ed efficienti.
Ulteriori motivi di interesse sono state le osservazioni circa i costrutti che costituiscono lo zoccolo teorico di qualsiasi linguaggio di programmazione come Java per l'implementazione di routine concorrenti.
È evidente che si tratta di un argomento attuale e dalle applicazioni numerosissime; non solo nel campo dei sistemi operativi ma anche nelle pure applicazioni orientate alla rete in generale ed a Internet in particolare.
Nei prossimi appuntamenti vedremo altri aspetti dei sistemi operativi, cercando come sempre di svelare i metodi usati e approfondendo gli algoritmi di gestione degni di studio.

Fabio Grimaldi

Biblioteca

ON LINE

BOING BOING

Per la seconda volta blog dell'anno. Boing Boing non ha "completamente" a che fare con la programmazione, ma è un sito interessante per tutti coloro che vogliono scoprire dove vanno le nuove tecnologie, cosa succede di interessante su Internet, a cosa stanno studiando i ricercatori. Infine Boing Boing contiene curiosità notizie e commenti un po' su tutto, postato in modo originale e ricco di link.

http://boingboing.net/

KURO5HIN

Una valida alternativa a Slahdot.org. Kuro5hin è un "journal" di cultura "geek". Geek è un termine che viene utilizzato in modo gergale per indicare uno studioso, una persona che non fa sport o che è particolarmente attratta dalle materie scientifiche piuttosto che da quelle letterarie.

Geek è normalmente chi vive immerso nella tecnologia, chi riempie interi fogli di calcoli matematici, e quasi sempre un programmatore. Kuro5Hin mantiene fede alle sue promesse, è aggiornato, completo e interessante.

http://www.kuro5hin.org

L'ARTE DELL'HACKING

LE IDEE, GLI STRUMENTI, LE TECNICHE DEGLI HACKER

Un libro coraggioso questo. Coraggioso perché tiene fede al suo titolo. Non si tratta del classico abuso del termine "Hacker", piuttosto il libro si addentra nelle tecniche usate dagli hacker in modo pratico e con una serie di esempi decisamente arditi. D'altra parte Hacker è colui che cerca di scoprire i propri limiti e li abbatte proponendosi un nuovo obiettivo. E' proprio questo che insegna il libro, ovvero come muoversi lungo la sottile linea di confine che separa un programmatore "normale" da un "hacker". All'interno troverete la descrizioni di tecniche per la realizzazione di Exploit, imparerete come sfruttare il buffer overflow, come portare un attacco DoS, come craccare le password. La cosa che lascia stupefatti è che queste tecniche sono corredate di codice di esempio completo e ben documentato. Decisamente da comprare se siete minimamente interessati a scoprire quali sono i vostri limiti, a viaggiare in universo sconosciuto e per lo più affascinante, insomma a diventare un Hacker.

Difficoltà: Alta • **Autore:** Jon Erickson • **Editore:** Apogeo • **ISBN:** 88-503-2280-1 • **Anno di pubblicazione:** 2004 • **Lingua:** Italiana • **Pagine:** 237 • **Prezzo:** € 24,00

MANUALE PRATICO DI JAVA

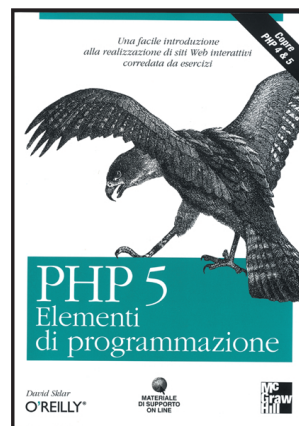
LA PROGRAMMAZIONE DELLA PIATTAFORMA J2EE

Java è un linguaggio completo che consente di costruire applicazioni potenti, sufficientemente veloci, e soprattutto multipiattaforma. Molti desiderano approdare a Java, riconoscendone le qualità e le caratteristiche uniche di portabilità e completezza. Questo "manuale pratico di Java" si pone come uno strumento intermedio, che da un lato non ha la "banalità/semplicità" delle guide dedicate ai principianti, dall'altro mantiene una forma di esposizione e una linearità tali da renderlo un sicuro riferimento sia per chi inizia a ha bisogno di conferme come di indicazioni, sia per chi è già un esperto e ha bisogno di approfondire alcuni temi. Il libro è stato scritto da un pool di autori tutti italiani, il che è già di per se sinonimo di sensibilità verso un modo di intendere la programmazione tipico del nostro paese, non è dunque viziato da forzature di traduzione e contiene chiari riferimenti a quelle che sono le esigenze di un programmatore che agisce nel nostro mercato.

Difficoltà: media • **Autore:** P. Aiello, M. Bigatti, L. Cerquetti, A. Giovannini, G. Morello, G. Puliti, S. Rossini, N. Venditti • **Editore:** Hops/Tecniche Nuove • **ISBN:** 88-481-1582-9 • **Anno di pubblicazione:** 2004 • **Lingua:** Italiana • **Pagine:** 664 • **Prezzo:** € 39,90

PHP 5

ELEMENTI DI PROGRAMMAZIONE

PHP5 è stato rilasciato ormai da quasi un anno. La nuova versione del linguaggio contiene significative modifiche, prime fra tutte un migliorato supporto alla programmazione ad oggetti. Anche l'integrazione con MySQL, l'uso delle estensioni e altre parti importanti del linguaggio hanno subito variazioni piuttosto importanti. Nonostante questo PHP5 stenta a rimpiazzare il già collaudatissimo PHP4. Questo libro edito da O'RELLY per la firma di David Sklar si pone come guida rapida all'uso del linguaggio. Per "guida pratica" non si intende scarsità di informazione, piuttosto il libro si propone come essenziale, rapido da consultare e affronta gli argomenti importanti senza scendere in dettagli che non sono fondamentali per programmare in PHP. E' un libro da comprare se avete intenzione di imparare il PHP e volete farlo iniziando direttamente da PHP5. Gli argomenti affrontati vanno dai primi passi fino all'uso di tecniche complesse quali ad esempio la gestione di file XML.

Difficoltà: bassa • **Autore:** David Sklar • **Editore:** O'Reilly • **ISBN:** 88 386 4420-9 • **Anno di pubblicazione:** 2005 • **Lingua:** Italiana • **Pagine:** 355 • **Prezzo:** € 30,00